N° d'ordre

THESE

présentée à

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"Informatique"

par

Xavier de LAMBERTERIE

—o—

ESPACES VIRTUELS ET GESTION DE FICHIERS

—o—

Thèse soutenue le 13 Juin 1973 devant la commission d'examen

Monsieur N. GASTINEL Président

Messieurs L. BOLLIET
M. GRIFFITHS
C. HANS
} Examinateurs

N° d'ordre

THESE

présentée à

L'UNIVERSITE SCIENTIFIQUE ET MEDICALE DE GRENOBLE

pour obtenir

LE GRADE DE DOCTEUR DE TROISIEME CYCLE

"Informatique"

par

Xavier de LAMBERTERIE

—o—

ESPACES VIRTUELS ET GESTION DE FICHIERS

—o—

Thèse soutenue le 13 Juin 1973 devant la commission d'examen

Monsieur N. GASTINEL Président

Messieurs L. BOLLIET
M. GRIFFITHS
C. HANS
} Examinateurs

UNIV.
LEIDEN
BIBL.

UNIVERSITE SCIENTIFIQUE
ET MEDICALE DE GRENOBLE

LISTE DES PROFESSEURS

Président : Monsieur Michel SOUTIF
Vice-Président : Monsieur Gabriel CAU

PROFESSEURS TITULAIRES

MM.	ANGLES D'AURIAC Paul	Mécanique des fluides
	ARNAUD Georges	Clinique des maladies infectieuses
	ARNAUD Paul	Chimie
	AYANT Yves	Physique approfondie
Mme	BARBIER Marie-Jeanne	Electrochimie
MM.	BARBIER Jean-Claude	Physique expérimentale
	BARBIER Reynold	Géologie appliquée
	BARJON Robert	Physique nucléaire
	BARNOUD Fernand	Biosynthèse de la cellulose
	BARRA Jean-René	Statistiques
	BARRIE Joseph	Clinique chirurgicale
	BENOIT Jean	Radioélectricité
	BESSON Jean	Electrochimie
	BEZES Henri	Chirurgie générale
	BLAMBERT Maurice	Mathématiques Pures
	BOLLIET Louis	Informatique (IUT B)
	BONNET Georges	Electrotechnique
	BONNET Jean-Louis	Clinique ophtalmologique
	BONNET-EYMARD Joseph	Pathologie médicale
	BONNIER Etienne	Electrochimie Electrométallurgie
	BOUCHERLE André	Chimie et Toxicologie
	BOUCHEZ Robert	Physique nucléaire
	BRAVARD Yves	Géographie
	BRISSONNEAU Pierre	Physique du Solide
	BUYLE-BODIN Maurice	Electronique
	CABANAC Jean	Pathologie chirurgicale
	CABANEL Guy	Clinique rhumatologique et hydrologie
	CALAS François	Anatomie
	CARRAZ Gilbert	Biologie animale et pharmacodynamie
	CAU Gabriel	Médecine légale et Toxicologie
	CAUQUIS Georges	Chimie organique
	CHABAUTY Claude	Mathématiques Pures
	CHARACHON Robert	Oto-Rhino-Laryngologie
	CHATEAU Robert	Thérapeutique
	CHENE Marcel	Chimie papetière
	COEUR André	Pharmacie chimique
	CONTAMIN Robert	Clinique gynécologique
	COUDERC Pierre	Anatomie Pathologique
	CRAYA Antoine	Mécanique
Mme	DEBELMAS Anne-Marie	Matière médicale
MM.	DEBELMAS Jacques	Géologie générale
	DEGRANGE Charles	Zoologie
	DESSAUX Georges	Physiologie animale
	DODU Jacques	Mécanique appliquée
	DREYFUS Bernard	Thermodynamique
	DUCROS Pierre	Cristallographie
	DUGOIS Pierre	Clinique de Dermatologie et Syphiligraphie
	FAU René	Clinique neuro-psychiatrique
	FELICI Noël	Electrostatique
	GAGNAIRE Didier	Chimie physique
	GALLISSOT François	Mathématiques Pures
	GALVANI Octave	Mathématiques Pures

MM.	GASTINEL Noël	Analyse numérique
	GERBER Robert	Mathématiques Pures
	GIRAUD Pierre	Géologie
	KLEIN Joseph	Mathématiques Pures
Mme	KOFLER Lucie	Botanique et Physiologie végétale
MM.	KOSZUL Jean-Louis	Mathématiques Pures
	KRAVTCHENKO Julien	Mécanique
	KUNTZMANN Jean	Mathématiques Appliquées
	LACAZE Albert	Thermodynamique
	LACHARME Jean	Biologie végétale
	LATREILLE René	Chirurgie générale
	LATURAZE Jean	Biochimie pharmaceutique
	LAURENT Pierre	Mathématiques Appliquées
	LEDRU Jean	Clinique médicale B
	LLIBOUTRY Louis	Géophysique
	LOUP Jean	Géographie
Mle	LUTZ Elisabeth	Mathématiques Pures
	MALGRANGE Bernard	Mathématiques Pures
	MALINAS Yves	Clinique obstétricale
	MARTIN-NOEL Pierre	Seméiologie médicale
	MASSEPORT Jean	Géographie
	MAZARE Yves	Clinique médicale A
	MICHEL Robert	Minéralogie et Pétrographie
	MOURIQUAND Claude	Histologie
	MOUSSA André	Chimie nucléaire
	NEEL Louis	Physique du Solide
	OZENDA Paul	Botanique
	PAUTHENET René	Electrotechnique
	PAYAN Jean-Jacques	Mathématiques Pures
	PEBAY-PEYROULA Jean-Claude	Physique
	PERRET René	Servomécanismes
	PILLET Emile	Physique industrielle
	RASSAT André	Chimie systématique
	RENARD Michel	Thermodynamique
	REULOS René	Physique industrielle
	RINALDI Renaud	Physique
	ROGET Jean	Clinique de pédiatrie et de puériculture
	SANTON Lucien	Mécanique
	SEIGNEURIN Raymond	Microbiologie et Hygiène
	SENGEL Philippe	Zoologie
	SILBERT Robert	Mécanique des fluides
	SOUTIF Michel	Physique générale
	TANCHE Maurice	Physiologie
	TRAYNARD Philippe	Chimie générale
	VAILLAND François	Zoologie
	VAUQUOIS Bernard	Calcul électronique
Mme	VERAIN Alice	Pharmacie galénique
M.	VERAIN André	Physique
Mme	VEYRET Germaine	Géographie
MM.	VEYRET Paul	Géographie
	VIGNAIS Pierre	Biochimie médicale
	YOCCOZ Jean	Physique nucléaire théorique

PROFESSEURS ASSOCIES

MM.	BULLEMER Bernhard	Physique
	RADHAKRISHNA Pidatala	Thermodynamique

PROFESSEURS SANS CHAIRE

MM.	AUBERT Guy	Physique
	BEAUDOING André	Pédiatrie
	BERTRANDIAS Jean-Paul	Mathématiques Appliquées
	BIARES Jean-Pierre	Mécanique
	BONNETAIN Lucien	Chimie minérale
Mme	BONNIER Jane	Chimie générale
MM.	CARLIER Georges	Biologie végétale
	COHEN Joseph	Electrotechnique
	COUMES André	Radioélectricité
	DEPASSEL Roger	Mécanique des Fluides
	DEPORTES Charles	Chimie minérale
	DESRE Pierre	Métallurgie
	DOLIQUE Jean-Michel	Physique des Plasmas
	GAUTHIER Yves	Sciences biologiques
	GEINDRE Michel	Electroradiologie
	GIDON Paul	Géologie et Minéralogie
	GLENAT René	Chimie organique
	HACQUES Gérard	Calcul numérique
	JANIN Bernard	Géographie
Mme	KAHANE Josette	Physique
MM.	MULLER Jean-Michel	Thérapeutique
	PERRIAUX Jean-Jacques	Géologie et minéralogie
	POULOUJADOFF Michel	Electrotechnique
	REBECQ Jacques	Biologie (CUS)
	REVOL Michel	Urologie
	REYMOND Jean-Charles	Chirurgie générale
	ROBERT André	Chimie papetière
	SARRAZIN Roger	Anatomie et chirurgie
	SARROT-REYNAULD Jean	Géologie
	SIBILLE Robert	Construction Mécanique
	SIROT Louis	Chirurgie générale
Mme	SOUTIF Jeanne	Physique générale
M.	VALENTIN Jacques	Physique nucléaire

MAITRES DE CONFERENCES ET MAITRES DE CONFERENCES AGREGES

Mle	AGNIUS-DELORD Claudine	Physique pharmaceutique
	ALARY Josette	Chimie analytique
MM.	AMBLARD Pierre	Dermatologie
	AMBROISE-THOMAS Pierre	Parasitologie
	ARMAND Yves	Chimie
	BEGUIN Claude	Chimie organique
	BELORIZKY Elie	Physique
	BENZAKEN Claude	Mathématiques Appliquées
Mme	BERTRANDIAS Françoise	Mathématiques Pures
MM.	BLIMAN Samuel	Electronique (EIE)
	BLOCH Daniel	Electrotechnique
Mme	BOUCHE Liane	Mathématiques (CUS)
MM.	BOUCHET Yves	Anatomie
	BOUSSARD Jean-Claude	Mathématiques Appliquées
	BOUVARD Maurice	Mécanique des Fluides
	BRIERE Georges	Physique expérimentale
	BRODEAU François	Mathématiques (IUT B)
	BRUGEL Lucien	Energétique
	BUISSON Roger	Physique
	BUTEL Jean	Orthopédie
	CHAMBAZ Edmond	Biochimie médicale
	CHAMPETIER Jean	Anatomie et organogénèse

MM. CHIAVERINA Jean	Biologie appliquée (EFP)
CHIBON Pierre	Biologie animale
COHEN-ADDAD Jean-Pierre	Spectrométrie physique
COLOMB Maurice	Biochimie médicale
CONTE René	Physique
CROUZET Guy	Radiologie
DURAND Francis	Métallurgie
DUSSAUD René	Mathématiques (CUS)
Mme ETERRADOSSI Jacqueline	Physiologie
MM. FAURE Jacques	Médecine légale
GAVEND Michel	Pharmacologie
GENSAC Pierre	Botanique
GERMAIN Jean-Pierre	Mécanique
GIDON Maurice	Géologie
GRIFFITHS Michaël	Mathématiques Appliquées
GROULADE Joseph	Biochimie médicale
HOLLARD Daniel	Hématologie
HUGONOT Robert	Hygiène et Médecine préventive
IDELMAN Simon	Physiologie animale
IVANES Marcel	Electricité
JALBERT Pierre	Histologie
JOLY Jean-René	Mathématiques Pures
JOUBERT Jean-Claude	Physique du Solide
JULLIEN Pierre	Mathématiques Pures
KAHANE André	Physique générale
KUHN Gérard	Physique
Mme LAJZEROWICZ Jeannine	Physique
MM. LAJZEROWICZ Joseph	Physique
LANCIA Roland	Physique atomique
LE JUNTER Noël	Electronique
LEROY Philippe	Mathématiques
LOISEAUX Jean-Marie	Physique Nucléaire
LONGEQUEUE Jean-Pierre	Physique Nucléaire
LUU DUC Cuong	Chimie Organique
MACHE Régis	Physiologie végétale
MAGNIN Robert	Hygiène et Médecine préventive
MARECHAL Jean	Mécanique
MARTIN-BOUYER Michel	Chimie (CUS)
MAYNARD Roger	Physique du Solide
MICOUD Max	Maladies infectieuses
MOREAU René	Hydraulique (INP)
NEGRE Robert	Mécanique
PARAMELLE Bernard	Pneumologie
PECCOUD François	Analyse (IUT B)
PEFFEN René	Métallurgie
PELMONT Jean	Physiologie animale
PERRET Jean	Neurologie
PERRIN Louis	Pathologie expérimentale
PFISTER Jean-Claude	Physique du Solide
PHELIP Xavier	Rhumatologie
Mle PIERY Yvette	Biologie animale
MM. RACHAIL Michel	Médecine interne
RACINET Claude	Gynécologie et obstétrique
RICHARD Lucien	Botanique
Mme RINAUDO Marguerite	Chimie macromoléculaire
MM. ROMIER Guy	Mathématiques (IUT B)
ROUGEMONT (DE) Jacques	Neuro-Chirurgie
STIEGLITZ Paul	Anesthésiologie

MM. STOEBNER Pierre	Anatomie pathologique
VAN CUTSEM Bernard	Mathématiques Appliquées
VEILLON Gérard	Mathématiques Appliquées (INP)
VIALON Pierre	Géologie
VOOG Robert	Médecine interne
VROUSSOS Constantin	Radiologie
ZADWORNY François	Electronique

MAITRES DE CONFERENCES ASSOCIES

MM. BOUDOURIS Georges	Radioélectricité
CHEEKE John	Thermodynamique
GOLDSCHMIDT Hubert	Mathématiques
YACOUD Mahmoud	Médecine légale

CHARGES DE FONCTIONS DE MATIRES DE CONFERENCES

Mme BERIEL Hélène	Physiologie
Mme RENAUDET Jacqueline	Microbiologie

Fait le 8 MARS 1972.

Digitized by the Internet Archive
in 2026

https://archive.org/details/IA41534920_0020

Je tiens à remercier :

Monsieur le Professeur Noël GASTINEL, Directeur du Laboratoire de Calcul qui m'a fait l'honneur de présider le Jury de cette thèse et qui a accordé un vif intérêt à mon travail.

Monsieur Louis BOLLIET, Professeur à l'Institut Universitaire de Technologie de Grenoble qui m'a encouragé dans mes travaux.

Monsieur Michaël GRIFFITHS, Maître de Conférence à l'Université Scientifique et Médicale de Grenoble, qui s'est intéressé à mon travail et a accepté de faire partie du Jury.

Monsieur Claude HANS du Centre Scientifique IBM de Grenoble qui m'a proposé de travailler sur le sujet CP+/CMS+ et m'a guidé tout au long de mes recherches.

Tous mes collègues du Laboratoire de Calcul et du Centre Scientifique pour leur aide efficace et leurs critiques constructives, en particulier Messieurs AUROUX, BELLINO, DUPUY, LE HEIGET, JACOLIN.

Je tiens enfin à remercier les services de tirage pour la réalisation matérielle de cet ouvrage.

RESUME DE LA THESE

Nous avons participé à la conception et réalisé en partie une gestion de fichiers pour le système CP+/CMS+ (H1) basée sur un emploi systématique des mécanismes de pagination.

Cette gestion de fichiers est construite sous forme de niveaux indépendants qui, partant de l'espace physique disponible le structure en espaces logiques multiples. On a fait disparaitre dès les premiers niveaux les caractéristiques des unités physiques. La structure d'un fichier se présente sous la forme d'une vaste mémoire virtuelle pouvant en général résider sur des unités physiques multiples et de types différents. Comme la capacité d'adressage de la machine employée (24 bits sur le 360/67) est insuffisante pour pouvoir ranger un fichier dans une mémoire virtuelle gérée par hardware, nous avons introduit un mécanisme de fenêtre pour accéder à un sous ensemble d'un fichier.

C H A P I T R E I

1) HOMOGENEISATION DES UNITES PERIPHERIQUES REELLES

De nombreuses applications en informatique (gestion de listes de personnel, gestion de stocks, banques, bibliothèques, etc...) manipulent de grandes quantités de données. Les utilisateurs souhaitent donc que les systèmes d'exploitation mettent à leur disposition des moyens efficaces pour stocker et pour retrouver ces données afin de les consulter ou de les mettre à jour.

Habituellement le stockage se fait sur des mémoires secondaires (disques, tambours) et il est nécessaire de structurer les données. Montrons deux méthodes possibles de structuration. La première consiste à découper l'information en enregistrements logiques ordonnés qui constituent des fichiers. La deuxième consiste à adopter une structure en banques de données, c'est à dire à exprimer des liens entre objets de nature différente. Par exemple une bibliothèque regroupe des auteurs, des livres, des publications etc... et on veut exprimer le fait que tel auteur a écrit tels livres. La consultation de la banque de données se fait alors à l'aide d'un langage évolué. On peut par exemple demander la liste

des livres écrits par tel auteur, les publications traitant d'un même sujet, etc.. (A1). Dans la suite de la thèse nous ne nous intéresserons qu'à la première forme de stockage c'est à dire aux fichiers.

Les deux principaux problèmes d'une gestion de fichiers sont d'une part de donner des moyens efficaces pour accéder aux enregistrements, et d'autre part de rendre très simple l'emploi des unités périphériques (disques, tambours). On constate que les systèmes de la deuxième génération (par exemple OS) mettent à la disposition des utilisateurs des méthodes d'accès évoluées. Ainsi l'accès aux enregistrements peut se faire de façon séquentielle c'est à dire retrouver les enregistrements dans l'ordre de leur création, de façon directe c'est à dire accéder au nième enregistrement sans lire les n-1 qui précèdent, et enfin de façon séquentielle indéxée. Dans cette organisation les enregistrements sont ordonnés non pas d'après l'ordre de leur création, mais par des "clés" qui les identifient. Les clés sont par exemple une portion d'un enregistrement. Par exemple dans un fichier de pièces détachées on peut prendre comme clé le numéro de série de chaque pièce et ces numéros ordonnent le fichier. Il est possible d'insérer des enregistrements, d'en enlever etc.. car la position physique des enregistrements n'intervient pas.

On constate par contre que l'emploi des unités périphériques est complexe car ces systèmes les organisent d'après la structure de l'unité périphérique elle-même. Il en résulte que chaque unité a une gestion propre. D'autre part la gestion de leurs contenus se fait statiquement. Ainsi la création d'un fichier sous OS par exemple, impose de calculer sa taille maximum, de choisir une unité périphérique de résidence, d'en connaitre les caractéristiques et de déterminer la place occupée sur cette unité. En cas d'erreur, si la

place allouée n'est pas assez grande il faut en ajouter et la probabilité de trouver de l'espace contigu au précédent est très faible. Il est donc nécessaire de transférer le fichier à un autre emplacement.

Avec le développement des mécanismes de pagination d'une part et des systèmes conversationnels d'autre part il est apparu des gestions de fichiers basées sur des notions nouvelles. Une des premières est l'allocation véritablement dynamique d'espace sur mémoire secondaire. En effet l'utilisateur est devenu plus exigeant car il est interactif depuis un terminal (machine à écrire). Il veut disposer d'un espace de mémoire secondaire pour y ranger ses propres fichiers et demander à tout instant de les modifier (édition), d'en créer, d'en effacer. Une gestion trop statique de l'espace secondaire s'avère impossible. Elle conduit rapidement à une saturation de la place disponible ce qui impose des réorganisations constantes.

Une autre notion est l'indépendance entre la gestion de fichier et les unités périphériques. En effet les systèmes conversationnels ont permis de faire participer un nombre croissant d'utilisateurs sans que ceux-ci aient besoin de connaitre le fonctionnement interne du système. La manipulation des fichiers doit être la plus simple possible : l'utilisateur connait l'aspect externe de son fichier, donc l'aspect logique, et le système se charge de le représenter physiquement.

Parmis les systèmes conversationnels actuels, CP/CMS (I1,I2), développé initialement à CAMBRIDGE, fait partie d'une classe particulière. En effet le concept fondamental de CP est la génération de machines virtuelles (A2) répliques de machines réelles en partageant les ressources d'un calculateur réel (360/67) (I3) entre plusieurs utilisateurs. Ainsi chacun d'eux depuis un terminal

(machine à écrire) a l'illusion d'avoir à sa disposition une machine complète. Il peut y charger le système de son choix, et on a pu activer sous CP des systèmes tels que: TSS, MTS, CMS, OS, CP etc...

En général les utilisateurs emploient CMS qui est un système conversationnel mono-utilisateur, CP assurant la partie partage de temps. La gestion de fichiers de CMS est simple d'emploi car c'est elle qui gère le disque virtuel de l'utilisateur, et lui donne des moyens d'accéder aux fichiers. L'utilisateur peut faire abstraction de l'unité virtuelle qu'il emploie. Cependant un disque virtuel sous CP est par définition un groupe de cylindres contigus sur une unité réelle. Ceci entraine des limitations: un disque virtuel ne peut être plus grand qu'un disque réel. D'autre part la gestion des disques réels pose le même problème que sous OS, mais au niveau de l'ensemble des fichiers. En effet lorsqu'un utilisateur demande la création d'une nouvelle machine virtuelle, il doit préciser la place globale que va occuper ses fichiers pour déterminer la taille du disque virtuel dont il a besoin. De même si l'espace alloué s'avère insuffisant il faut l'étendre et la probabilité de trouver de l'espace contigu au disque virtuel alloué est très faible. Il faut donc transférer l'ensemble du disque virtuel à un autre emplacement.

Nous avons donc résolu dans le cadre du projet CP+/CMS+ (H1) le problème suivant: donner à l'utilisateur un moyen d'accès simple aux unités périphériques sans qu'il connaisse leur structure,et même la possibilité d'adresser de façon continue des unités physiquement hétérogènes. L'unité virtuel de l'utilisateur peut résider sur plusieurs unités physiques de différentes natures. Au point de vue taille, on peut adresser à l'heure actuelle de façon continue environ 80.000 disques réels ce qui peut être considéré comme infini.

La solution adoptée est de structurer les unités réelles d'après la structure de la machine employée et non pas d'après la structure de l'unité elle-même. Par exemple si on découpe les unités réelles en blocs identiques de taille t, l'accès à toutes les unités se fera de la même façon. Le système CP emploie des mécanismes de pagination pour gérer les mémoires virtuelles. La mémoire réelle et la mémoire virtuelle sont donc structurées en pages (4 kilo octets). Si on fixe la taille t précédente aussi à la page, on se trouve en présence de trois ensembles (mémoire réelle, mémoire virtuelle, unités périphériques) qui sont composés d'éléments de même nature. Les échanges entre ces ensembles sont simples, et on peut employer les mécanismes de pagination pour éxécuter tous les transferts de pages.

Les unités périphériques se présentent à l'utilisateur sous forme d'une suite de pages contiguës adressables par leurs numéros. L'utilisateur ignore où résident ces pages et lorsqu'il veut éxécuter une entrée-sortie, il le demande à CP. Celui-ci détermine sur quelles unités réelles sont placées les pages de l'utilisateur concerné et éxécute l'entrée sortie à l'aide des mécanismes de pagination.

Les principaux avantages sont: un accès aux périphériques simplifié (la notion de programme canal a disparu), et une gestion améliorée des périphériques réelles. En effet les pages pouvant résider sur des unités hétérogènes, il est possible d'en ajouter à un utilisateur sans modifier celles qui lui sont déjà allouées. En contre-partie, la machine virtuelle qui utilise ces mécanismes n'est plus la réplique d'une machine réelle, car elle utilise directement des fonctions de CP. On l'appelle donc environnement car elle ne pourra être activé que sous CP. Les suites de pages alloués aux utilisateurs constituent des unités périphériques logiques qu'on appelle 1-ESPACE (chapitre 2).

Le système CMS+ n'étant pas suffisamment développé, nous avons décidé dans une première étape d'appliquer ces mécanismes à CMS pour les mettre au point et étudier leur influence sur le comportement global du système CP/CMS.

2) LA GESTION DE FICHIERS DE CMS+

CMS+, destiné à remplacer CMS, est par définition un système activé dans un environnement. Nous avons étudié sa gestion de fichiers qui utilise les espaces homogènes définis ci-dessus. Celle-ci est organisée en niveaux hiérarchiques (D4).

Un des aspects les plus importants des niveaux hierarchiques est que chacun constitue un tout logique et ne communique qu'avec le niveau immédiatement inférieur.

Cette technique présente de nombreux avantages. La description d'un ensemble complexe peut se faire en définissant les fonctions dont on a besoin niveau par niveau. Ceci simplifie la conception, la programmation et la mise au point. En effet les niveaux peuvent être implémentés successivement, et mis au point les uns après les autres par un nombre de tests limités aux fonctions du niveau considéré. Lorsque le niveau N est satisfaisant on peut réaliser le niveau N+1. Au fur et à mesure qu'on monte dans la hiérarchie, l'ensemble offre plus de possibilités, mais la mise au point reste simple car en cas d'erreurs, on détermine à quel niveau elles se produisent et on y remedie. Un autre avantage est l'indépendance entre niveaux. En effet il est possible de modifier les niveaux inférieurs sans changer les niveaux supérieurs, c'est à dire d'adapter l'ensemble sur tel ou tel système avec le minimum de modifications, et dans le cas d'une gestion de fichier, d'être indépendant des unités périphériques.

Pour la gestion de fichiers de CMS+ on s'est inspiré de l'article de S.E. MADNICK (M1) qui propose une organisations en six niveaux. Le fichier est considéré comme une mémoire virtuelle et les differents niveaux traduisent les demandes logiques de l'utilisateur en ordres physiques. Rappelons brièvement le fonctionnement des niveaux.

Le sixième niveau qui est le plus élevé donc le plus abstrait, consiste a découper la mémoire qui représente le fichier en enregistrements, c'est à dire fait apparaitre les différentes méthodes d'accès : séquentiel, direct, fixe, variable, indexes, etc... C'est donc un interface entre l'utilisateur et le fichier qui est propre au système considéré (macro OS, plist de CMS ..).

Le cinquième niveau considère le fichier au niveau logique, c'est à dire en tant que mémoire virtuelle repérée par un nom symbolique. Il est muni de routines qui peuvent adresser directement un caractère dans le fichier (accéder au nième octet du fichier de nom A).

Le quatrième niveau considère le fichier de façon physique. A ce niveau on sait que le fichier n'est pas une mémoire virtuelle, mais est découpé en plusieurs blocs physiques. Il apparait la notion de descripteur de fichier, d'emplacement physique des enregistrements. D'autre part on vérifie aussi à ce niveau les droits d'accès et on résoud les problèmes de partage. Ceci permet un partage des fichiers à un niveau trés fin : l'enregistrement ou le groupe d'enregistrements. Il est donc possible de partager un fichier simultanément en lecture écriture entre plusieurs utilisateurs.

Le troisième niveau fait apparaitre la notion d'unité périphérique. Il est chargé de grouper les différents enregistrements dans des blocs dont la taille est adaptée à un périphérique donné (formattage des unités à accès direct). D'autre

part ce niveau gère les mémoires tampons utilisées pour les entrées sorties. Par exemple lors de la lecture d'un enregistrement, la totalité du bloc physique contenant l'enregistrement est lue en mémoire, mais seul l'enregistrement est fourni au niveau supérieur.

Le deuxième niveau est chargé d'optimiser les entrées sorties sur un périphérique donné. Cela se traduit par une optimisation des mouvements de bras d'un disque, du nombre de révolution d'un tambour ('slot-sorting').

Enfin le premier niveau est constitué du superviseur d'entrée sortie du système considéré. C'est lui qui gère les files d'attente d'entrées sorties sur les périphériques, reçoit toutes les interruptions, et traite les erreurs.

Une telle hiérarchie est trés générale et la plupart des gestions de fichiers ont pu y être représentées. Dans le cas de CMS+ on l'a adaptée au concept d'environnement. En effet les deux premiers niveaux décrits ci-dessus sont à la charge de CP+ puisqu'un environnement lui demande d'éxécuter les entrées sorties réelles. Remarquons aussi que le deuxième niveau ne peut être traité par une machine virtuelle si plusieurs utilisateurs ont des disques virtuels sur un même disque réel. En effet si l'un d'eux optimise les mouvements du bras de son disque, l'activation par CP+ d'un autre utilisateur peut repositionner le bras de façon arbitraire et détruire l'optimisation du premier. CP+ est donc le seul qui puisse optimiser efficacement.

Le troisième niveau constitue un interface entre l'environnement et les unités réelles c'est à dire CP+. Cela revient à munir CP+ de fonctions d'entrée sortie accessibles aux environnements, ou encore de leurs attacher des unités périphériques logiques définies précédement, qui constituent le premier niveau de la gestion de fichiers de CMS+.

Le deuxième niveau de la gestion de fichiers de CMS+ appelé 2-ESPACE (chapitre 4), décrit le fichier sous forme d'une mémoire virtuelle (2**31 bytes). Un 1-ESPACE est donc découpé en sous-ensembles de pages qui constituent les 2-ESPACE. Il contient en outre un catalogue qui décrit ceux-ci. On ne considère pas la partie description comme un niveau car d'une part on laisse de coté les problèmes de partage au niveau du fichier dans cette première approche, et d'autre part le fait d'utiliser comme unité de transfert la page rend uniforme la description des fichiers quelles que soient les unités sur lesquels ils résident. Accéder à un enregistrement revient à déterminer dans quelle page du 1-ESPACE il se trouve à l'aide du catalogue et d'amener cette page en mémoire.

Enfin le troisième niveau de la gestion de fichiers regroupe les différentes méthodes d'accès de CMS+. Ces dernières, n'étant pas suffisamment développées, ne seront pas étudiées dans la suite de la thèse.

Avant d'aborder les 2-ESPACE, on présente les modifications qu'on a été amené à faire à la gestion de la mémoire virtuelle par CP (chapitre 3). En effet celle-ci est définie statiquement par CP. Il est intéressant de la rendre dynamique en utilisant la segmentation du 360/67. Ceci est possible grâce au concept d'environnement qui permet d'exploiter les possibilités de CP depuis une machine virtuelle.

C H A P I T R E 2

1 - E S P A C E S

Dans cette partie nous montrons la réalisation d'un mécanisme qui permet de rendre homogène à l'utilisateur des unités périphériques physiquement hétérogènes. La solution adoptée consiste à structurer les unités périphériques en pages et de faire communiquer CP avec les machines virtuelles (appelées alors environnements) pour éxécuter les entrées-sorties. Ces dernières sont éxécutées par la pagination de CP.

1) CONCEPT DE 1-ESPACE.

1.1) Définition.

Dans CP un 1-ESPACE est une suite ordonnée de N+1 pages logiques (bloc de 4 K octets (K = 1024)) numérotées (0,1,2... N). Ces pages résident en permanence sur des unités physiques de types divers (disques, tambours). Des pages logiquement contiguës du 1-ESPACE ne sont pas nécessairement physiquement contiguës.

1.2) Caractéristiques.

1.2.1. Protection

CP partitionne l'ensemble de l'espace mémoire secondaire disponible entre les différentes machines virtuelles et environnements. La place destinée à l'ensemble des environnements doit aussi être partitionnée en sous-espaces disjoints. En effet différents groupes d'utilisateurs souhaitent ne pas avoir de communications entre eux (projets concurrents). C'est parce que les sous-espaces créés sont effectivement disjoints qu'aucune intervention d'un groupe sur l'autre ne peut avoir lieu. Cette notion fondamentale justifie la présence de 1-ESPACES multiples sous un même système CP. Il sera possible de moduler à loisir en fonction des impératifs de chaque installation le nombre de 1-ESPACE (depuis un seul couvrant l'ensemble de l'espace disponible, jusqu'à l'autre extrême soit plusieurs 1-ESPACES par utilisateur).

1.2.2) Indépendance vis-à-vis des caractéristiques physiques des unités.

On souhaite faire disparaître toute dépendance vis-à-vis des caractéristiques des unités physiques. Ainsi lorsqu'on parle d'un disque, on s'exprime en terme de nombre de cylindres, nombre de pistes, nombre d'enregistrements par piste etc... Ces paramètres dépendent d'un matériel particulier et sont sujets à modification (à la suite de la disponibilité de nouveaux matériels, ou de la disparition d'ancien). Ces paramètres sont en outre multiples puisque sur une configuration donnée on trouve en général plusieurs

types d'unités connectées.

Nous souhaitons au contraire, ainsi que l'implique la définition, qu'un 1-ESPACE puisse résider sur des unités multiples qui ne soient pas nécessairement de mêmes types. Un autre paramètre limitatif lié à l'emploi d'unité physique est leur capacité : nous désirons nous affranchir aussi de cette contrainte et potentiellement pouvoir définir un 1-ESPACE comme illimité. Pour ce faire nous avons décidé de nommer la suite de pages constituant un 1-ESPACE à l'aide d'une adresse de 31 bits. Ceci correspond à plus de 2.10**9 pages de 4 K soit plus de 8.10**12 caractères, soit quelques 80.000 disques de 100 millions de caractères chacun.

1.2.3) Simplicité

L'ensemble des adresses qui servent à nommer les pages du 1-ESPACE est la suite des entiers positifs ou nuls : 0, 1, 2, 3... n. Cet adressage est beaucoup plus simple à manipuler que des adresses de type classique CC, HH, R (numéro de cylindre, de tête, d'enregistrement). En outre l'adressage est continu c'est-à-dire que la page qui suit la pième a pour adresse p+1 alors que celle qui suit CC, HH, R est rarement CC, HH, R+1.

On veut en outre, pour lire ou écrire une page, pouvoir le demander à l'aide d'une fonction primitive très simple qui n'implique pas la construction d'une chaine de CCW (Chanel Command Word) et un ou plusieurs SIO. Pour ce faire on utilise le fait que CP existe et on lui demande de réaliser une lecture ou une écriture d'une façon aussi simple que possible (à l'aide de l'instruction DIAGNOSE (I2, L1)).

1.2.4) Homogénéité

Parce qu'on a choisi comme unité de transfert (ou atome) entre le 1-ESPACE et la mémoire virtuelle la page, il est possible d'utiliser toutes les fonctions de pagination de CP qui de longue date ont été optimisées pour être efficaces.

Rappelons brièvement le fonctionnement de la pagination de CP. Ce dernier utilise deux sortes de tables: les premières sont imposées par le hardware du 360/67, les secondes sont propres à CP.

Les 24 bits d'une adresse virtuelle sont découpées en trois groupes:

a) les 4 bits de poids fort constituent le numéro de segment.

b) les 8 bits de poids moyen constituent le numéro de page.

c) les 12 bits de poids faible constituent un déplacement dans une page. Les tables imposées par le hardware sont:

- la table des 16 segments qui contient pour chaque entrée soit l'adresse d'une table de pages ainsi que le nombre de pages du segment, soit un bit d'invalidité s'il n'y a aucune page pour ce segment.

- les tables de pages (256 entrées au maximum) qui contiennent le numéro de la page réelle où réside une page virtuelle, ou un bit d'invalidité si la page virtuelle n'est pas en mémoire réelle.

La traduction dynamique des adresse indexe successivement la table de segments, puis la table de pages obtenue. CP dispose en outre d'une table de pages externes qui decrit les pages virtuelles qui ne résident pas en mémoire réelle. Toutes ces tables sont construites à l'initialisation de la machine virtuelle (LOGIN).

Lire une page d'un 1-ESPACE se limite à modifier une entrée de la table de pages externes. Ceci diffère l'opération effective

d'entrée-sortie jusqu'à ce que la page soit effectivement référencée. La mémoire virtuelle joue donc le rôle d'une fenêtre sur le 1-ESPACE que l'utilisateur peut déplacer facilement.

Cette opération de lecture s'éffectue donc sans entrée-sortie. Il n'y a pas de transfert d'information. La page du 1-ESPACE est intégrée ou encore incluse dans la mémoire virtuelle en modifiant la configuration de cette dernière.

1.2.5) Croissance

Comme on l'a vu, la taille potentielle d'un 1-ESPACE est pratiquement infinie. Mais en fait on n'alloue à chaque utilisateur que l'espace physique dont il a besoin (par exemple 1000 pages). Lorsque cet espace est plein, il est possible d'accroître la place attribuée au 1-ESPACE sans que ceci implique des modifications du contenu de la partie déjà existante de ce 1-ESPACE.

1.2.6) Accès

Le 1-ESPACE est représenté sous CP comme une unité périphérique au même titre qu'un disque, une console etc... et se référence à l'aide d'une adresse hexadécimale de type CUU (Canal, Unité de Controle, Unité).

Un 1-ESPACE peut globalement être accessible par un seul utilisateur ou, au contraire être partagé par plusieurs. Il peut être globalement déclaré en lecture seulement ou en lecture écriture. Cette derniére propriété n'exclut pas qu'à l'intérieur du 1-ESPACE il y ait des zones accessibles avec des privilèges plus restrictifs (fichiers en lecture seulement dans un 1-ESPACE en lecture-écriture).

1.3) Relations avec la notion de mini-disque.

1.3.1) Concept

La notion de 1-ESPACE est une généralisation de celle de mini-disque. Un mini-disque est défini comme une suite de cylindres contigus apparaissant sur un même volume (cette définition limite la taille d'un mini-disque à la taille d'un disque physique).

Le 1-ESPACE se présente sous forme de groupes de pages contigues appelés 1-SEGMENT. Les groupes par contre ne sont pas contigus et n'apparaissent pas nécessairement sur les mêmes volumes, ni sur les mêmes types de volume.

Lorsque le 1-ESPACE se réduit à un 1-SEGMENT et que ce dernier réside sur un disque on retrouve sensiblement la notion de mini-disque : seule la méthode d'adressage diffère. Par contre, dans le cas de 1-SEGMENTS multiples, un 1-ESPACE peut recouvrir plusieurs disques physiques.

Il est enfin possible d'avoir tout ou partie d'un 1-ESPACE apparaissant sur un tambour, alors que les mini-disques ne résident que sur disque physique.

1.3.2) Protection

L'analogie est ici totale entre le mini-disque et le 1-ESPACE. Chacun fait l'objet de déclarations distinctes dans le catalogue de CP qui décrit les configurations des machines virtuelles et environnements. L'utilisateur d'un mini-disque ou d'un 1-ESPACE n'a aucun moyen de référencer un autre mini-disque ou 1-ESPACE ne lui

appartenant pas.

1.3.3) Modes d'Accès

On peut accéder au 1-ESPACE dans sa totalité :

- En lecture seulement par plusieurs utilisateurs (RDONLY).
- En lecture-écriture par un seul, ou en lecture seulement par plusieurs (RDSHARE).
- En lecture-écriture simultanément par plusieurs utilisateurs (cette option est sous le contrôle des niveaux logiques supérieurs qui sont chargés d'assurer la cohérence des informations stockées).

1.3.4) Opérations d'entrée-sortie

Le principe de fonctionnement est totalement différent. Un mini-disque se programme comme un disque réel à l'aide d'instructions machines, de programmes canaux et d'instructions d'entrée-sortie (CCW, SIO, (I4)). La machine virtuelle soumet le programme canal à CP par une instruction SIO et CP traduit le programme canal virtuel en programme canal réel (compilation des programmes canaux).

Par contre une opération d'entrée-sortie sur un 1-ESPACE se fait exclusivement à l'aide des routines de pagination de CP. Une opération de lecture consiste à inclure une page du 1-ESPACE dans la mémoire virtuelle en modifiant la table de pages externes. Il serait possible d'inclure tout un 1-ESPACE en mémoire virtuelle si celle-ci était assez grande. Comme ce n'est pas le cas sur le 360/67 on laisse le soin aux environnements d'inclure les pages du 1-ESPACE en mémoire virtuelle, quand ils en ont besoin, en invoquant CP (à l'aide de l'instruction DIAGNOSE). Plus précisemment ils donnent un

couple : numéro de page dans le 1-ESPACE, numéro de page en mémoire. CP modifie alors l'entrée adéquate de la table de pages externes. Cette opération a lieu sans entrée sortie et c'est la référence ultérieure à la page (lire, éxécuter, ou modifier) qui déclanchera le transfert vers la mémoire réelle.

Contrairement à certains systèmes (MULTICS, ESOPE) on ne laisse pas modifier un 1-ESPACE directement par la pagination. Ceci est du principalement à deux raisons:

a) On veut assurer la protection en lecture d'un 1-ESPACE sans modifier la pagination existante. Lorsque CP décide de recopier une page de la mémoire réelle sur support secondaire et que cette page a été lue depuis un 1-ESPACE, on le force à réécrire cette page ailleurs, car à ce niveau on n'a pas de moyen simple de savoir si le 1-ESPACE est en lecture seule ou non. En effet CP dispose d'un mécanisme permettant de savoir que cette page appartient à un 1-ESPACE, mais on ne sait pas auquel elle appartient.

b) On veut assurer un maximum de sécurité en cours de fonctionnement. En effet on veut être sûr qu'une page a été réécrite dans un 1-ESPACE et si on laisse ce soin à la pagination de CP on ne sait pas à quelle moment l'écriture a lieu.

C'est pour ces deux raisons qu'on laisse aux environnements le soin de demander l'écriture des pages sur un 1-ESPACE (à l'aide de l'instruction DIAGNOSE). Il est alors possible de tester si on a le privilège d'écrire ou non, et on est sûr aprés éxécution de cette fonction que la page est écrite.

Il est aussi possible de tester si une page lue depuis un 1-ESPACE a été modifiée ou non depuis son entrée en mémoire (utilisation du hardware du 360/67). On peut donc, dans certains cas, ne pas éxécuter l'écriture si on veut réécrire une page dans l'emplacement du 1-ESPACE d'où on l'a lue.

En résumé on voit donc que les entrées-sorties pour un 1-ESPACE sont par conception plus économe en temps CP et en entrées-sorties réelles (on ne lit que les pages dont on a effectivement besoin, et on détecte les cas où l'écriture est inutile).

1.4) Application dans l'espace physique.

L'objet de ce paragraphe est de définir comment à partir d'un numéro de page dans un 1-ESPACE on est en mesure de déterminer l'unité physique sur laquelle elle réside ainsi que son adresse sur cette unité.

La réalisation de la fonction qui applique l'ensemble des entiers positifs ou nuls (numéros de pages) dans l'ensemble constitué par des doublets (identification d'unité physique déplacement sur l'unité physique) peut être faite de diverses façons. C'est dire que la partie de CP qui réalise cette transformation d'adresse peut être facilement modifiée et qu'on pourrait substituer aisément à l'algorithme que nous proposons tout autre méthode jugée ultérieurement préférable.

Un 1-ESPACE est constitué d'une suite (S0, S1... Sn) de 1-SEGMENT. Le 1-SEGMENT Si est un ensemble de pages contiguës sur un même support physique, tandis que deux 1-SEGMENT logiquement consécutifs dans le 1-ESPACE peuvent ne pas être physiquement contigus dans l'espace réel. Un 1-SEGMENT est défini sur une unité physique par l'identification de l'unité, l'adresse de la première page qu'il comprend, sa longueur c'est-à-dire le nombre de pages qu'il contient, soit le triplet:

(Ui, Di, Li)

L'ensemble des triplets (Ui, Di, Li) pour i variant de 0 à n suffit à définir la transformation.

La pième page du 1-ESPACE étant recherchée, on détermine le 1-SEGMENT K tel que la relation soit satisfaite:

$$\sum_{i=-1,k-1} Li \leqslant P < \sum_{i=-1,k} Li \quad (Li=0 \text{ pour } i=-1)$$

Ayant ainsi déterminé le numéro K, on trouve la page Ap cherchée avec:

$$Ap=P-\sum_{i=-1,k-1} Li+Dk$$

Il suffit enfin connaissant le numéro Ap d'une page sur une unité physique particulière de le convertir en adresse reconnaissable par l'unité (CCHHR par exemple). Outre les triplets précisés précédemment, il suffit de mémoriser la taille c'est-à-dire le nombre de pages du 1-ESPACE de façon à rejeter d'entrée toute tentative d'adressage en dehors des limites du 1-ESPACE.

Remarquons qu'il est possible d'avoir des 1-SEGMENTS qui ne résident nulle part (Ui, Di, Ø), ce qui permet des discontinuités d'adressage dans le 1-ESPACE.

2) LE 1-ESPACE VU PAR L'UTILISATEUR.

2.1) Aspect externe.

L'utilisateur voit le 1-ESPACE sous la forme d'une suite de pages numérotées de 0 à N. Toute dépendance vis-à-vis des unités réelles a disparu et l'accès aux pages se fait de façon plus rapide grâce aux primitives de manipulation qui utilisent la pagination.

Remarquons que le 1-ESPACE constitue une unité purement 'logique' et que tout programme utilisateur employant des 1-ESPACES ne pourra être activé que sous CP.

D'autre part le fait d'employer des pages apporte quelques restrictions. En effet l'utilisateur doit fournir des adresses alignées sur frontière de page pour toutes lectures ou écritures. Il n'est donc pas possible de lire une page d'un 1-ESPACE à n'importe quelle adresse en mémoire. Ceci pose donc le problème de l'acquisition de mémoire alignée sur frontière de page.

2.2) Reconnaissance d'un 1-ESPACE.

L'utilisateur dispose d'une fonction :

TEST (AD,NB)

avec AD : adresse du 1-ESPACE (CUU)

NB : nombre de pages donné en retour.

Cette fonction lui permet de savoir si une unité est un 1-ESPACE ou non. Il précise l'adresse du 1-ESPACE en tant qu'unité et en retour il a soit un code erreur indiquant que l'unité n'est pas un 1-ESPACE, soit une réponse positive ainsi que le nombre de page du 1-ESPACE.

2.3) Lecture.

La lecture des pages se fait à l'aide de la fonction :

LIREPAGE (AD,AM,N,NB)

- AD adresse du 1-ESPACE (CUU)
- AM numéro de page en mémoire virtuelle
- N numéro de page dans le 1-ESPACE
- NB nombre de pages à lire

Cette fonction lit les pages indiquées en mémoire. Remarquons que NB désigne plusieurs pages contigues de numéro N, N+1 etc...

Ex : LIRE (198, 20, 8, 4)

Lire la page 8 en mémoire à l'adresse 20

9 21

10 22

11 23

2.4) Ecriture.

L'écriture des pages se fait à l'aide de la fonction :

ECRIREPAGE (AD,AM,N,NB)

Les paramètres sont identiques à LIREPAGE. Cette fonction écrit les pages indiquées dans le 1-ESPACE. C'est l'inverse de la fonction LIREPAGE.

2.5) Test des pages modifiées.

La fonction:

TESTPAGE (AD,AM,N)

permet de savoir si la page numéro N d'un 1-ESPACE qui a été lue en mémoire à l'adresse AM a été modifiée. Au retour CP donne un code indiquant que la page a été modifiée ou non.

Cette fonction sera employée par exemple par les 2-ESPACE (chapitre 4) pour optimiser la gestion du catalogue.

3) PRINCIPE DE REALISATION SOUS CP.

3.1) Définition dans le catalogue.

Les machines virtuelles sont définies par un fichier système appelé le catalogue. Les enregistrements utilisant le mot clé 'UNIT' définissent des unités virtuelles. Afin d'assurer le maximum de compatibilité avec les unités classiques (console, mini-disque etc...) définies par des enregistrements utilisant le mot clé 'UNIT', on a décidé de définir les 1-ESPACES en utilisant le même formalisme.

Comme on l'a vu un 1-ESPACE est une suite de 1-SEGMENT et fonctionnellement la définition d'un 1-SEGMENT peut se faire de la même façon qu'un mini-disque. On utilise donc autant d'enregistrements 'UNIT' que le 1-ESPACE comprend de 1-SEGMENT. Un type d'unité spécial (3333) est utilisé pour faire la distinction avec les autres unités, et une numérotation évite de permuter accidentellement les 1-SEGMENTS.

```
Ex : 001,F003   UNIT  198,3333,CPDSK4,050,054
     002        UNIT  198,3333,CPDSK2,002,008
     003        UNIT  198,3333,CPDSK4,020,025
```

Ceci définit un 1-ESPACE contenant 3 1-SEGMENT, d'adresse 198 (CUU).

Le premier enregistrement est privilégié. Il contient le nombre de 1-SEGMENT (F003) ainsi que les paramètres de partage éventuels (RDSHARE, RDONLY etc...). Les enregistrements suivants contiennent de l'information redondante (CUU, 3333) pour assurer une plus grande sécurité.

On remarque que l'allocation est faite par cylindres contigus et non au niveau des pages pour faciliter la gestion des disques réels.

Il est possible de définir des extensions vides (discontinuités d'adressage) en utilisant comme nom de volume : EMPTY.

3.2) Activation d'un 1-ESPACE par CP.

L'activation d'un 1-ESPACE se fait à l'aide de blocs descriptifs en mémoire réélle, et de primitives de manipulation décrites au paragraphe suivant.

Au 'LOGIN' d'un utilisateur CP lit le catalogue, et construit les blocs associés aux unités virtuelles.

Rappelons brièvement comment sont décrites les unités réelles et virtuelles (cf. fig. 2.1):

- les canaux sont décrits par des blocs (RCHBLOK, VCHBLOK, R pour réel, V pour virtuel), chainés entre eux, qui contiennent les adresses des blocs décrivant les unités de contrôle rattachées au canal.

- les unités de contrôle sont décrites de même à l'aide de blocs (RCUBLOK, VCUBLOK) chainés entre eux qui contiennent l'adresse des blocs décrivant les unités connectées.

- les unités sont décrites par des blocs (RDEVBLOK, VDEVBLOK).

Le lien entre les unités virtuelles et réelles se fait au niveau des VDEVBLOK et des RDEVBLOK et indique sur quelle unité réelle se trouve une unité virtuelle.

La description des 1-ESPACE se fait de la même façon qu'une unité. Il a une adresse (CUU) et il est représenté par les trois blocs : VCHBLOK, VCUBLOK, VDEVBLOK. La différence essentielle est qu'un 1-ESPACE peut résider sur plusieurs unités réelles. Le lien unique qui existe entre un VDEVBLOK et un RDEVBLOK a été modifié. On a inclu entre les deux une table supplémentaire (FTABLE) qui sert d'aiguillage entre le VDEVBLOK et les RDEVBLOK où résident les

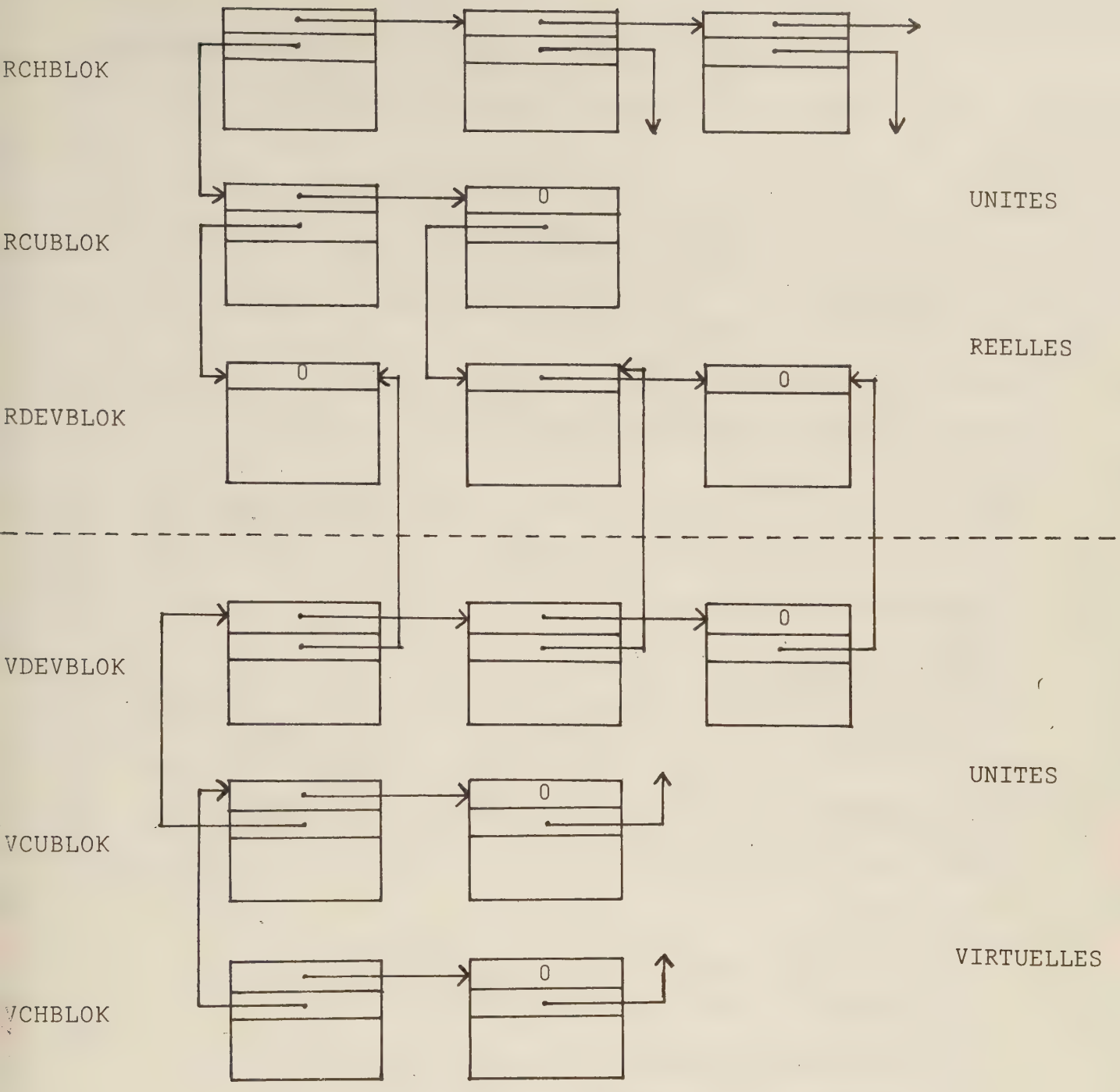
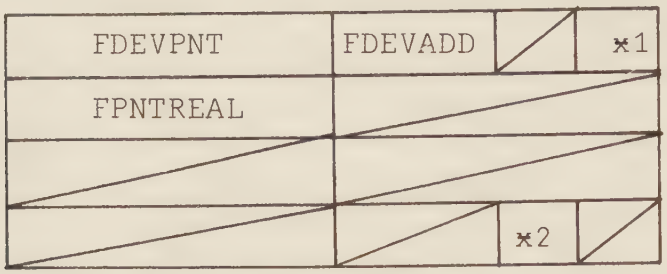


Figure 2.1 Unités réelles et virtuelles

différents 1-SEGMENT.

Il a été nécessaire de modifiér légèrement le VDEVBLOK lorsqu'il décrit un 1-ESPACE on l'appelle FDEVBLOK. Sa structure est la suivante :



FDEVPNT : pointe vers le VDEVBLOK ou FDEVBLOK suivant

FDEVADD : CUU (adresse du 1-ESPACE)

*1 FDEVTYPE : type '3333'

FPNTREAL : adresse de la FTABLE

*2 FDEVFLG : unité en lecture, lecture-écriture, temporaire.

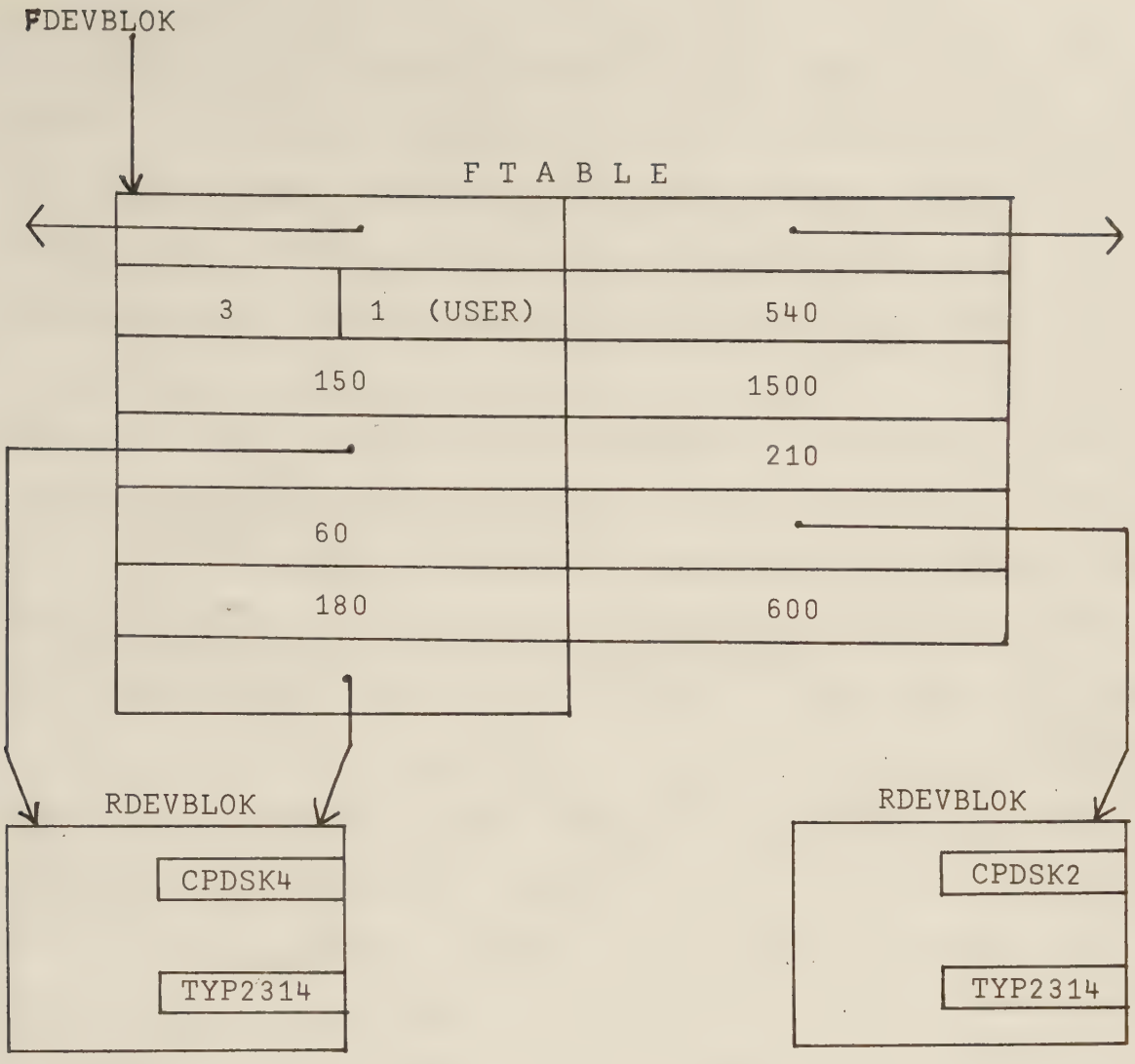
Son format reste donc compatible avec le VDEVBLOK. La FTABLE est structurée d'après les triplets de définition d'un 1-ESPACE (cf 1.4). Elle contient (cf fig. 2.2):

a) un en-tête

- pointeur vers la FTABLE suivante,
- pointeur vers la FTABLE précédente,
- nombre de 1-SEGMENT,
- nombre d'utilisateurs,
- nombre de pages.

b) autant de triplet que de 1-SEGMENT

- nombre de pages du 1-SEGMENT,
- numéro réél de la première page du 1-SEGMENT sur l'unité physique,
- pointeur vers le RDEVBLOK.



1 utilisateur
3 1-SEGMENT
540 pages (150+210+180)

Figure 2.2 FTABLE obtenue d'après l'exemple du §3.1
(disque de type 2314, 30 pages par cylindre)

L'en-tête contient des informations globales comme le nombre de 1-SEGMENT et le nombre de pages. De plus il contient le nombre d'utilisateurs ainsi que des pointeurs chainant les FTABLE entre elles. Ceci évite de dupliquer en mémoire réélle des FTABLE décrivant le même 1-ESPACE si celui-ci est employé par plusieurs utilisateurs.

Ceci nous amène aux problèmes de partage. Ces derniers s'appliquent aux 1-ESPACE dans leur totalité. On interdit de définir des 1-ESPACE ayant un ou plusieurs 1-SEGMENT en commun.

Par définition : 'Deux 1-ESPACE ayant les mêmes extensions sont identiques' ou encore 'Deux 1-ESPACE décrits par la même FTABLE sont identiques'.

Un 1-ESPACE partagé est décrit dans chaque machine virtuelle par un VDEVBLOK qui contient les privilèges d'accès, la FTABLE par contre, est unique. Lorsqu'un utilisateur demande l'accès à un 1-ESPACE (au LOGIN en début de session, ou par la commande LINK en cours de session), on en recherche l'existence dans le catalogue. Ensuite on construit la FTABLE associée. Enfin on recherche s'il existe une FTABLE identique à celle construite. Si ce n'est pas le cas l'utilisateur est le seul à s'en servir et il accède au 1-ESPACE conformément aux paramètres d'accès du catalogue.

Si par contre on trouve une FTABLE identique, alors plusieurs utilisateurs ont accès au même 1-ESPACE. A l'aide du catalogue on résoud les problèmes de partage (accès possible, mot de passe, etc ...). Enfin la FTABLE construite au début est libéré, et on augmente le nombre d'utilisateurs de l'autre si l'accès est autorisé. Cette méthode s'avère plus efficace que dans le cas des mini-disques. En effet on ne parcourt que la liste des FTABLE, alors que pour les

mini-disques on parcourt la liste de toutes les unités de tous les utilisateurs.

En cours de fonctionnement il est nécessaire de se protéger des utilisateurs qui accéderaient aux 1-ESPACE sans passer par les primitives de manipulations. En effet les 1-ESPACE ne peuvent être programmés à l'aide des instructions machine. Ainsi toutes les instructions SIO, TIO etc... référençant des 1-ESPACE sont rejetés ainsi que la fonction console IPL.

3.3) Primitives de manipulation.

3.3.1 Remarques concernant la pagination

Le fait d'utiliser les routines de pagination pour accéder aux 1-ESPACE peut les rendre assimilables à des espaces de pagination d'un type particulier. On distingue donc :

-a) un espace de pagination banalisé, utilisé au gré de CP pour les mémoires virtuelles;

-b) des espaces de pagination appartenant aux utilisateurs (1-ESPACE)

Les tables de pages externes (SWAPTABLE pour CP) qui donnent l'adresse des pages lorsqu'elles ne sont pas en mémoire, doivent indiquer le type d'espace de pagination. Chaque entrée comprend un indicateur (appelé RECOMPUTE) qui permet de faire la distinction. En effet, CP n'est pas autorisé à 'paginer' directement dans un espace appartenant à un utilisateur. Seul les primitives de manipulation peuvent modifier les espaces de pagination de type 1-ESPACE.

Dans la suite de la description, on emploiera les termes 'espace de pagination' pour 'espace de pagination banalisé réservé à CP'.

3.3.2 Forme générale des primitives

L'accès aux 1-ESPACES se fait par l'instruction DIAGNOSE dont le format est :

DIAG RI,D(RJ)

- DIAG désigne le code (X'83')
- D permet de distinguer plusieurs classes de diagnose (X'28' ici)
- RI désigne un registre qui contient :
  - un code (TEST, LIREPAGE, ECRIREPAGE, TESTPAGE)
  - l'adresse du 1-ESPACE (CUU)
- RJ désigne plusieurs registres suivant la fonction.

Cette instruction habituellement réservée aux services de maintenance est une instruction privilégiée. Son éxécution par les machines virtuelles et environnements en mode programme, provoque une interruption. CP reprend le contrôle et sélectionne à l'aide du code D une routine qu'il active.

Au retour un code signale les erreurs éventuelles.

3.3.3 Primitive TEST

Cette primitive permet de tester si une unité est de type 1-ESPACE. CP parcourt les blocs virtuels (VCHBLOK, VCUBLOK, VDEVBLOK) jusqu'à ce qu'il trouve l'unité CUU (dans RI). S'il ne la trouve pas un code erreur est retourné à l'utilisateur (R15).

Si l'unité trouvée est un 1-ESPACE la fonction donne dans RJ le nombre de pages du 1-ESPACE. Sinon elle renvoie un code d'erreur.

3.3.4 Primitive LIREPAGE

Nous rappelons que la lecture d'une page du 1-ESPACE se fait en l'incluant (cf 1.2.4) dans la mémoire virtuelle en modifiant la tables de pages externes qui décrit cette dernière. Il n'y a donc pas d'entrée-sortie réelle.

Cette primitive permet de lire des pages d'un 1-ESPACE en mémoire virtuelle. RJ désigne ici deux registres consécutifs.

- RJ qui contient le numéro de la page dans le 1-ESPACE
- RJ+1 qui contient le numéro de page en mémoire virtuelle, et le nombre de page à transferer

A l'éxécution on teste d'abord la validité des paramètres. S'ils sont incorrects on retourne un code d'erreur à l'utilisateur.

L'opération de lecture indique que l'état actuel de la page dans laquelle on va transférer l'information, n'intéresse plus l'utilisateur. On peut donc libérer cette page en mémoire réélle et sur l'espace de pagination. Etudions les deux cas:

a) en mémoire réélle

Le numéro de page dans Rj+1 permet de sélectionner une entrée de PAGETABLE et de SWAPTABLE. Si la page est en mémoire réélle (bit dans PAGETABLE) on libère cet emplacement.

b) Sur l'espace de pagination

Si la page a été réécrite dans l'espace de pagination on libère son emplacement. En effet l'allocation des pages dans cet espace se fait dynamiquement en cours de fonctionnement, et elles ne peuvent pas être conservées par l'utilisateur.

A l'aide de la FTABLE on traduit le numéro de page dans RJ en un doublet :

(RDEVCODE, numéro de page sur une unité réélle).

En fonction des caractéristiques de l'unité réélle, on transforme ce doublet en un autre :

(RDEVCODE, CCHHR)

qui est placé dans la SWAPTABLE (ainsi que le bit RECOMPUTE). La fonction est alors terminée.

Le nombre de pages à lire dans RJ+1 n'est valable que pour des pages logiquement contigues :

Ex : Lire les pages 10, 11, 12, 13, 14, 15

RJ=10

RJ+1=AM,6

Les pages sont lues en mémoire à l'adresse AM, AM+1 etc...

3.3.5 Primitive ECRIREPAGE

Contrairement à la fonction de lecture on ne peut pas écrire des pages dans un 1-ESPACE sans aucune entrée-sortie. Il est possible d'en minimiser le nombre dans certains cas.

Cette primitive permet d'écrire des pages de la mémoire virtuelle dans un 1-ESPACE. Le numéro RJ désigne aussi deux registres consécutifs qui ont la même signification que pour LIREPAGE.

A l'éxécution on commence là aussi par tester la validité des paramètres.

L'adresse mémoire dans RJ+1 permet de sélectionner une entrée de PAGETABLE et de SWAPTABLE.

A l'aide de la FTABLE, on traduit le numéro de page contenu dans RJ en un doublet :

(RDEVCODE, numéro de page sur une unité réelle)

puis en (RDEVCODE, CCHHR)

Deux cas se présentent :

a) L'entrée de la SWAPTABLE est égale à (RDEVCODE, CCHHR). Ceci signifie que la SWAPTABLE désigne déjà la page du 1-ESPACE où l'on veut écrire.

Dans ce cas si la page n'est pas en mémoire réelle, elle n'a jamais été modifiée depuis la précédente fonction LIRE. L'écriture est alors inutile.

Si la page est en mémoire réelle, on regarde si elle a été modifiée (clés du 360/67 (I4)). Si elle n'est pas modifiée l'écriture est encore inutile. Sinon on force la réécriture de la page dans le 1-ESPACE.

b) L'entrée de la SWAPTABLE n'est pas égale à (RDEVCODE, CCHHR)

Dans ce cas, on amène la page en mémoire si nécessaire, et on force sa réécriture dans le 1-ESPACE.

Si la page avait été recopiée dans l'espace de pagination, on libère cet emplacement car il est inutile de disposer de deux copies identiques de cette page sur support secondaire.

3.3.6) Primitive TESTPAGE.

Cette primitive permet de tester si une page lue sur un 1-ESPACE a été modifiée depuis son entrée en mémoire.

Les paramètres sont identiques à la primitive LIREPAGE (le nombre de page dans Rj+1 est obligatoirement 1).

Le déroulement de la primitive se passe comme pour la primitive ECRIREPAGE. Lorsqu'on arrive à la conclusion qu'il n'est pas nécessaire d'écrire la page on envoie un code indiquant que la page n'a pas été modifiée. Dans le cas inverse si l'écriture est nécessaire, elle n'est pas éxécutée et on envoie un code indiquant

que la page a été modifiée.

3.4) Désactivation d'un 1-ESPACE.

Lorsqu'un utilisateur quitte le système, ou utilise la commande 'DETACH', il est nécessaire de libérer les blocs de la mémoire réelle utilisés par le 1-ESPACE concerné.

Il y a en fait deux sortes de blocs à libérer :

- les blocs décrivant le 1-ESPACE,
- les blocs décrivant les entrées-sorties réelles en cours.

La libération des blocs décrivant l'unité (VCHBLOK, VCUBLOK, VDEVBLOK) se fait comme pour les autres unités virtuelles. Pour la FTABLE on diminue de 1 le nombre d'utilisateurs. Si ce nombre n'est pas nul on ne fait rien de plus. Si ce nombre est nul on libère l'espace mémoire occupé par la FTABLE et on réorganise la chaine.

La libération des blocs concernant les entrées-sorties réelles est plus simple que pour les mini-disques. Pour ces derniers il est nécessaire d'attendre la fin des entrées-sorties en cours avant de libérer les blocs décrivant l'unité virtuelle. Dans le cas des 1-ESPACE les entrées-sorties sont faites par la pagination de CP. Il est alors possible de libérer les blocs de description sans se préoccuper des entrées-sorties en cours. En effet lorsque CP reçoit une interruption pour fin d'opération de pagination, il est capable de la traiter même si l'utilisateur à qui elle est destinée a quitté le système, ou n'en a plus besoin.

4) UN EXEMPLE D'EMPLOI: LE S-ESPACE DE CMS.

Après réalisation des 1-ESPACE sous CP, on a décidé de les tester à l'aide de CMS. Il fallait donc définir dans celui-ci un 1-ESPACE dont l'utilisation soit transparente aux utilisateurs. De cette façon tous les utilisateurs de CMS emploieraient les mécanismes d'accès à ce 1-ESPACE ce qui permettrait de faire des tests d'une part, mais aussi d'étudier le comportement global du système CP/CMS lorsque ces mécanismes sont employés de façon intensive.

4.1 Rappels sur la gestion de fichiers de CMS.

Dans le système CMS les fichiers sont constitués de suites d'enregistrements (longueur fixe ou variable) repérés par un triplet:

NOM TYPE MODE

Le nom est une chaine de 8 caractères quelconques. Le type donne une indication sur le contenu du fichier.

Ex : Type ALGOL, FORTRAN, PL/1 etc...

Remarquons cependant que le type peut aussi être une chaine de 8 caractères quelconques au gré de l'utilisateur car aucun test n'est fait à la création d'un fichier.

Le mode est constitué de deux caractères :

identification d'unité, numéro de classe de fichier

Sous CMS il est possible d'accéder à plusieurs disques virtuels en même temps. C'est la commande ACCESS qui associe à une unité virtuelle une identification d'unité. On dit alors que l'unité est connectée à CMS. Les identifications possibles sont:

P - disque permanent

T - disque temporaire

A -

B -

S - disque système

C -

Le numéro de classe sert à sélectionner des groupes de fichiers sur une même unité et à leur attribuer des privilèges (lecture-écriture, lecture seulement etc ...).

Lors d'une recherche d'un fichier le mode permet de sélectionner une unité. S'il n'est pas précisé CMS explore toutes les unités connectées dans l'ordre : P, T, A, B, S, C.

Parmi les types de fichiers, il y en a deux particuliers :

- EXEC qui désigne des macro-commandes directement éxécutables
- MODULE qui désigne un fichier représentant une copie d'un programme sous la forme d'une image mémoire. Un tel fichier est donc directement chargeable (dans le sens de transferer son contenu en mémoire) et éxécutable. Toutes les commandes de CMS sont des fichiers de type module.

Lorsqu'un utilisateur envoie depuis son terminal la commande:

A

CMS recherche un fichier A EXEC, en parcourant tous les disques connectés, et lui donne le contrôle s'il le trouve. Sinon il recherche une commande de nom A résidente dans le noyau de CMS en mémoire. S'il ne la trouve pas il cherche un fichier A MODULE en reparcourant tous les disques. Si la commande nexiste pas CMS renvoie un message d'erreur.

On constate que la recherche d'une commande système est relativement longue, mais en contre-partie l'utilisateur peut redéfinir n'importe quelle commande en créant un fichier de type

EXEC ou MODULE.

4.2) Rappels sur le disque système de CMS.

4.2.1) Construction

Le disque système de CMS a une structure identique à celle d'un disque utilisateur (disque P). Sa construction se fait donc en l'attachant en tant que disque P à une machine virtuelle qui a le privilège d'y écrire.

Parmi tous les fichiers contenus dans cet espace, on en sélectionne un sous-ensemble en leur attribuant le mode 'P2'. Ce sous-ensemble constitue la partie du disque système accessible aux utilisateurs. En effet, lors d'un 'IPL' de CMS, (en employant la simulation de la fonction 'IPL' du système 360) pendant la phase d'initialisation, le système lit l'ensemble du catalogue contenu sur le disque système et isole dans une table particulière (SSTAT) les caractéristiques des fichiers dont le mode est 'P2'. Le catalogue tel qu'il avait été lu est oublié, seul subsiste la SSTAT qui vient d'être construite.

4.2.2) Emploi

Lorsque l'on recherche un fichier ayant pour mode 'SY', on regarde directement dans la table SSTAT. Ce cas est toutefois exceptionnel. En effet, comme on l'a vu précédemment, la recherche d'une commande se fait en explorant systématiquement tous les disques connectés. De façon analogue les fichiers systèmes qui ne sont pas des modules font l'objet du même mécanisme de recherche. Il s'agit en particulier de bibliothèques de 'MACRO' pour les

assembleurs (type 'MACLIB'), de bibliothèques de routines divers (type 'TXTLIB') etc...

La gestion du disque système pose des problèmes de performances. En effet il réside sur une seule unité physique (disque), et son emploi intensif peut provoquer des queues très longues sur l'unité de résidence.

Un autre aspect des performances provient du non emploi de la pagination (et du tambour) alors qu'on en dispose sous CP. En effet, un module sous CMS est une image de mémoire virtuelle, c'est-à-dire un ensemble de pages (dont deux au plus peuvent n'être que partiellement utilisées). En fait CMS ne tient pas compte de cette propriété et traite les modules comme les autres fichiers en les découpant en blocs de 800 caractères qui ne sont pas nécessairement physiquement contigus. La lecture d'un module se fait à l'aide d'entrée-sortie classique, c'est-à-dire construction de programmes canaux et 'SIO'.

4.3) Le S-ESPACE.

Comme on l'a vu un module de CMS représente un ensemble de pages. Il est donc possible de placer ces modules sur un 1-ESPACE sans remanier la gestion de fichier de CMS.

Par définition le S-ESPACE est un 1-ESPACE qui est composé en général de deux 1-SEGMENT : le premier réside sur tambour, le second sur disque. Il contient une partie des modules de CMS (les modules qui ne commencent pas sur une frontière de page ne peuvent pas y être placés de façon simple cf. 2.1).

Des mesures (R1) ont montré qu'on pouvait disposer de 50 pages sur tambour sans perturber la pagination, et d'autre part (L3) ces 50 pages suffisent à regrouper un sous-ensemble tel que 80% des

commandes émises soient sur le tambour.

Un module a donc trois lieux de résidence possible :

- sur le 1-SEGMENT tambour (modules très utilisés),
- sur le 1-SEGMENT disque (modules moins utilisés),
- sur le disque système (le reste).

En fait les modules dans le 1-ESPACE (appelés S-dule) sont conservés aussi sur le disque système pour des raisons de sécurité (défaillance possible du matériel).

4.4) Création du S-ESPACE.

Par création on entend ici 'remplissage' c'est-à-dire transférer effectivement des modules sur le S-ESPACE.

CP ne peut pas conserver d'information sur les tambours d'une session sur l'autre; en effet de par leur nature même les tambours sont des unités inamovibles et entre deux sessions de CP, un autre système (OS/MVT à Grenoble) peut les utiliser pour ses besoins propres.

Il est donc nécessaire de créer une image du S-ESPACE entièrement sur disque. De cette façon il est possible de le conserver d'une session à l'autre et un mécanisme décrit au paragraphe suivant permet de faire migrer la partie la plus employée sur tambour.

L'écriture des modules sur le S-ESPACE disque se fait à l'aide d'une nouvelle commande de CMS: 'GENDULE'. Seul l'administrateur du système peut s'en servir car il faut avoir le privilège d'écrire dans le S-ESPACE.

La commande 'GENDULE' consulte un fichier de CMS (CMS GENDULE) qui contient les noms des modules systèmes qu'il faut transférer sur le S-ESPACE disque. Dans un premier temps le module est lu en

mémoire. Il peut être placé à n'importe quelle adresse en mémoire, même si celle-ci diffère de l'adresse à partir de laquelle a été généré le module. En effet le module ne fait que transiter en mémoire et n'est pas éxécuté. Dans un deuxième temps l'ensemble des pages est écrit sur le S-ESPACE à l'aide de la primitive: ECRIREPAGE.

On a adopté une gestion très simple pour le S-ESPACE :

a) on le régénère entièrement en cas de modification;

b) la première page est réservée comme catalogue (S-bloc).

Les modules sont écrits les uns à la suite des autres sous forme de pages contigues. Pour chacun il suffit de se rappeler :

- le nom,
- le numéro de la page du S-ESPACE où il commence (NBPG1SSP),
- le nombre de page qu'il occupe (FTAILLE),
- l'adresse de chargement en mémoire (FENTRY),
- l'adresse du point d'entrée (FBEGIN),
- l'adresse de fin de module (FLOCCNT).

L'ensemble de ces informations constitue le S-bloc.

4.5) Transfert du S-ESPACE sur tambour.

Lors d'un IPL de CP le S-ESPACE disque est transféré, en partie ou en totalité sur tambour de façon automatique.

Les tambours sont réservés à la pagination de CP et le nombre de pages qu'on peut attribuer au S-ESPACE a été fixé expérimentalement (50 pages comme on l'a vu). Ce nouveau paramètre a été introduit parmi les constantes de description du système CP, donc est facilement modifiable.

La structure du S-ESPACE diffère suivant qu'il est entièrement sur disque, ou en deux 1-SEGMENT (tambour, disque). Il est donc

nécessaire après transfert de modifier la FTABLE initiale. En imposant que les pages à transférer soient en tête du S-ESPACE disque, la modification est simple.

Ex : S-ESPACE disque (CPDSK3, 500, 100) un 1-SEGMENT

après recopie des 50 premières pages :

(DRUM, 3, 50), (CPDSK3, 550, 50) deux 1-SEGMENT

(si la première page obtenue sur tambour porte le numéro 3).

Le transfert proprement dit ne peut être fait directement par CP. En effet la version utilisée ne lui permet pas d'employer les mécanismes de pagination pour son propre compte. Il faudrait redéfinir des fonctions spéciales. La solution adoptée consiste à faire éxécuter le transfert par le premier utilisateur du système : la machine virtuelle de l'opérateur dont le 'LOGIN' est automatique. Il est alors possible d'utiliser directement les primitives de manipulation (cf. 3.3).

Par convention on pose que le S-ESPACE est le 1-ESPACE d'adresse '19F' de l'opérateur. Si cette unité n'existe pas aucun transfert n'a lieu et le S-ESPACE n'existe pas.

Il suffit d'utiliser la primitive LIREPAGE pour amener dans la mémoire de l'opérateur les pages à transférer. On acquiert ensuite des pages sur tambour (contigues car on est les premiers à en demander) et on modifie la FTABLE initiale en lui adjoignant un 1-SEGMENT. Les pages sont écrites à l'aide de la primitive ECRIREPAGE.

Si les tambours sont indisponibles pour une raison quelconque la procédure de transfert n'a pas lieu. Il est possible d'utiliser le S-ESPACE sur disque. Nous bénéficions des mécanismes de pagination mais en cas d'usage intensif il peut y avoir saturation du canal disque. Les performances restent cependant aussi bonnes que si on utilise le disque système seulement.

4.6) Acquisation par l'utilisateur.

L'acquisition du S-ESPACE se fait à deux niveaux :

- en tant qu'unité virtuelle attachée à un environnement;
- en tant que S-ESPACE par CMS.

Du fait du transfert dynamique du S-ESPACE sur tambour il n'est pas possible de le définir dans le catalogue de description des machines virtuelles en utilisant le même formalisme que pour les 1-ESPACES (cf. 3.1). En effet à priori on ne sait pas sur quel tambour (s'il y en a plusieurs), et à partir de quelle page le S-ESPACE sera recopié. La définition du S-ESPACE en tant qu'unité dans le catalogue se fait à l'aide d'un enregistrement :

UNIT CUU, SSPACE

Ceci signifie qu'un utilisateur veut avoir accès au S-ESPACE. Il n'y a pas d'ambiguité possible, car tout utilisateur ne peut entrer dans le système qu'après l'opérateur, et dès que celui-ci est dans le système le S-ESPACE est parfaitement défini.

L'acquisition du S-ESPACE par CMS se fait automatiquement à l'IPL pendant la phase d'initialisation. Comme on l'a vu (cf. 4.2.1) CMS construit en mémoire le catalogue des fichiers système (SSTAT). Si on dispose du S-ESPACE (la fonction TEST l'indique), il faut amener en mémoire le S-bloc qui contient les informations nécessaires au chargement des commandes qui sont sur le tambour. On est en mesure de les charger à l'aide de la primitive LIREPAGE.

4.7) Emploi des S-dules.

L'emploi des S-dules est totalement transparent à l'utilisateur.

Lorsque CMS reçoit la commande 'A', il essaie de charger dans l'ordre un fichier A EXEC, une commande résidente de nom A, un fichier A MODULE dont la recherche se fait en parcourant tous les disques virtuels connectés (cf. 4.1). Le chargement d'un module se fait par la commande résidente 'LOADMOD' qui est la seule routine de CMS en dehors de l'initialisation qui a été modifiée.

La commande LOADMOD modifiée fonctionne de la façon suivante : elle commence par rechercher le nom du module dans le S-bloc. On voit donc que les S-dules sont assimilés à des commandes résidentes et qu'il n'est plus possible d'avoir sur son disque permanent (P) un module ayant le même nom qu'un S-dule (possibilité rarement employée). Il en résulte une recherche plus rapide en mémoire des commandes systèmes sans explorer toutes les unités connectées.

Si le S-dule de nom donné n'existe pas on laisse CMS éxécuter la procédure de chargement normale, c'est à dire éxécuter des entrées-sorties classiques par blocs de 800 caractères.

Si le S-dule existe on le charge en mémoire à l'aide des informations contenues dans le S-bloc et la primitive :

LIREPAGE (19F, FBEGIN, NBP61SSP, FTAILLE) (cf. 4.5)

D'autre part la constante FENTRY (adresse du point d'entrée) permet de donner le contrôle à la commande, et FLOCCNT (adresse de la fin du module) permet à CMS d'initialiser la mémoire libre sans perdre la place inoccupée dans la dernière page. On rappelle qu'aucune entrée-sortie réelle n'a lieu, et c'est seulement lorsque les pages sont référencées qu'elles sont chargées en mémoire par le mécanisme de pagination de CP. Ainsi on ne charge effectivement que la partie du module dont on a besoin.

5) MESURES.

5.1) Environnement de mesure.

Le S-ESPACE tel qu'il est implémenté est utilisé par tous les utilisateurs sans qu'ils le sachent. On peut donc affirmer que tout utilisateur de CMS emploie le S-ESPACE, ainsi que la primitive LIREPAGE chaque fois qu'il veut éxécuter une commande de CMS. Il en résulte une utilisation intensive des primitives d'accès au 1-ESPACE. On a donc mesuré le comportement du système pour voir quel est l'impact d'une telle modification et si l'utilisation des 1-ESPACE à grande échelle est rentable. On dispose aussi de mesures faites au moment où le S-ESPACE n'était pas encore en service. Il suffit de comparer les courbes pour observer les améliorations éventuelles.

Les mesures ont été prises à l'aide des mécanismes mis en place à l'I.M.A.G. (R1). L'état du S-ESPACE est le suivant:

1-SEGMENT tambour : 50 pages, soit 30 commandes de CMS.

1-SEGMENT disque : vide.

5.2) Aspect externe des mesures.

On entend par aspect externe ce que l'utilisateur peut voir comme amélioration depuis son terminal. Comme on l'a vu on ne peut que modifier le temps de chargement des commandes CMS, mais pas le temps d'éxécution. Ainsi lorsque le S-ESPACE n'est pas en fonction, on observe des temps de chargement:

pour FORTRAN de 0.19s

pour EDIT de 0.06s

etc...

Lorsque le S-ESPACE est en fonction on a:

pour FORTRAN 0.02s

pour EDIT 0.02s

etc...

On a donc des gains d'autant plus important que le nombre de pages de la commande est important (FORTRAN 16 pages, EDIT 5 pages). En faisant cette expérience pour toutes les commandes du S-ESPACE on obtient en moyenne un gain de chargement d'un facteur 4.

5.3) Aspect interne des mesures.

On entend par aspect interne le comportement global du système. Les mesures ont été faites au cours de deux périodes:

a) en septembre et octobre 72 où le S-ESPACE n'était pas en service. On a effectué 6105 mesures avec en moyenne 32,7 utilisateurs.

b) en février et mars 73 où le S-ESPACE était en service. On a effectué 5576 mesures avec en moyenne 31,7 utilisateurs.

Les courbes qui suivent sont exprimées en fonction du nombre d'utilisateurs connectés au système. Pour un nombre d'utilisateur donné, on n'a fait figurer que les points pour lesquels on a au moins 15 mesures. Les autres points sont considérés comme non significatifs.

On a mesuré quatre types de grandeur:

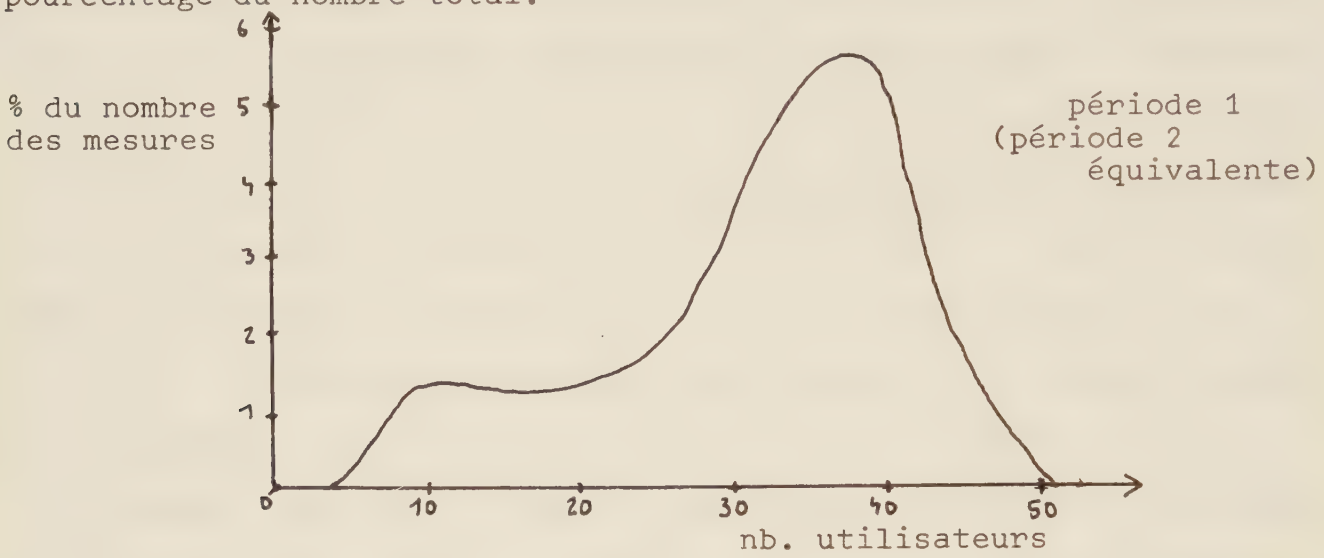
a) des mesures de temps qui sont exprimées en poucentage du temps total (temps système problème attente) (cf figures 2.11, 2.12, 2.13, 2.14).

b) des nombres d'entrées-sorties qui sont exprimés en nombre moyen par seconde (cf figures 2.5, 2.6).

c) des nombres d'entrées sorties en attente sur un périphérique exprimés en nombre moyen sur des intervalles de 100 secondes (cf figures 2.3, 2.4, 2.7, 2.8).

d) des nombres d'opération de pagination exprimés en nombre moyen de transferts de pages par secondes (cf figures 2.9, 2.10).

Ci-dessous figure la courbe de répartition qui pour un nombre d'utilisateurs donné, donne le nombre de mesures faites exprimées en pourcentage du nombre total.



5.3.1) Canaux disques.

Les courbes sur les figures 2.3, 2.4 montrent une nette baisse de la longueur des queues d'attente sur les disques. Ceci s'explique simplement par le fait que 80% des commandes sont lues sur le tambour. Les entrées-sorties qui restent sont celles faites sur les disques des utilisateurs et sur la partie du disque système qui n'est pas sur tambour.

5.3.2) Entrées-sorties virtuelles.

Les courbes des figures 2.5, 2.6 montrent aussi une nette baisse des entrées-sorties faites par les machines virtuelles. La

différence entre les courbes représente le nombre des entrées-sorties nécessaires au chargement des commandes qui est important comme on peut le constater.

5.3.3) Canal tambour.

Les courbes des figures 2.7, 2.8 montrent que l'attente sur le tambour n'a pas augmenté, et a même diminué. Ce résultat peut paraître surprenant, car la majorité des commandes de CMS y sont lues. En fait ceci s'explique car toute entrée-sortie sur disque s'accompagnent d'une ou plusieurs opérations de pagination. En effet CP doit amener et bloquer en mémoire réelle les pages virtuelles concernées par l'entrée-sortie.

Prenons comme exemple le chargement du module 'EDIT' (qui fait 5 pages):

a) CP va amener progressivement en mémoire réelle les 5 pages car à priori il ne sait pas que CMS va les remplir entièrement: soit 5 opérations de pagination.

b) CMS éxécute n entrées-sorties disque.

c) Si le système est trés chargé, il se peut que CP ait besoin des premières pages chargées en mémoire avant que l'éditeur prenne le contrôle. Il va donc les recopier sur le tambour pour les affecter à d'autres utilisateurs. Il y a donc en tout entre 5 et 10 opérations de pagination, et n entrées-sorties disque.

d) Enfin CMS donne le contrôle a 'EDIT'.

Si on utilise le S-ESPACE le chargement à l'aide de la primitive LIREPAGE est instantané et on arrive directement à la phase d). On économise donc à la fois des entrées-sorties disque, et des opérations de pagination.

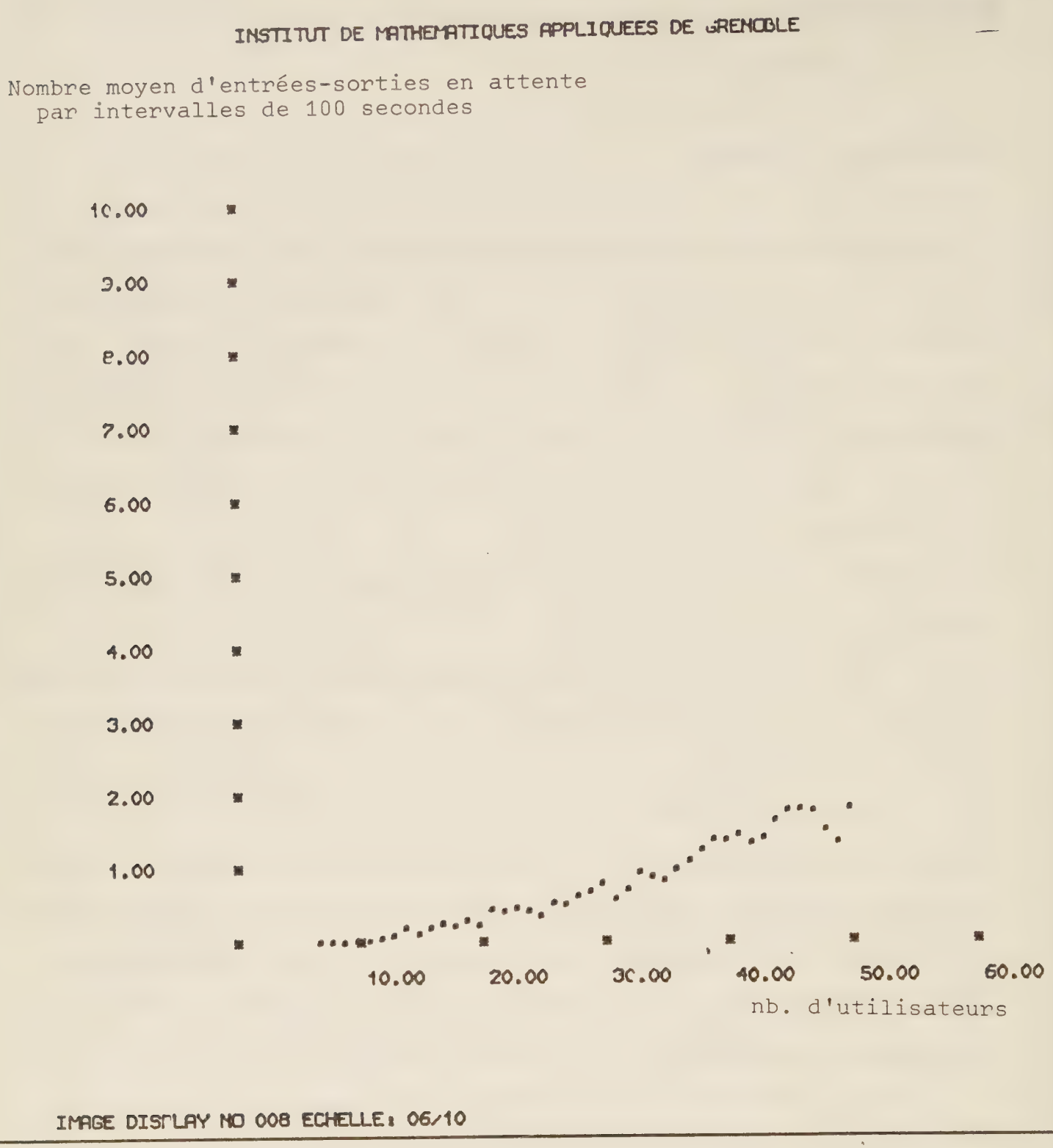


Figure 2.3 Queue disque sans S-ESPACE

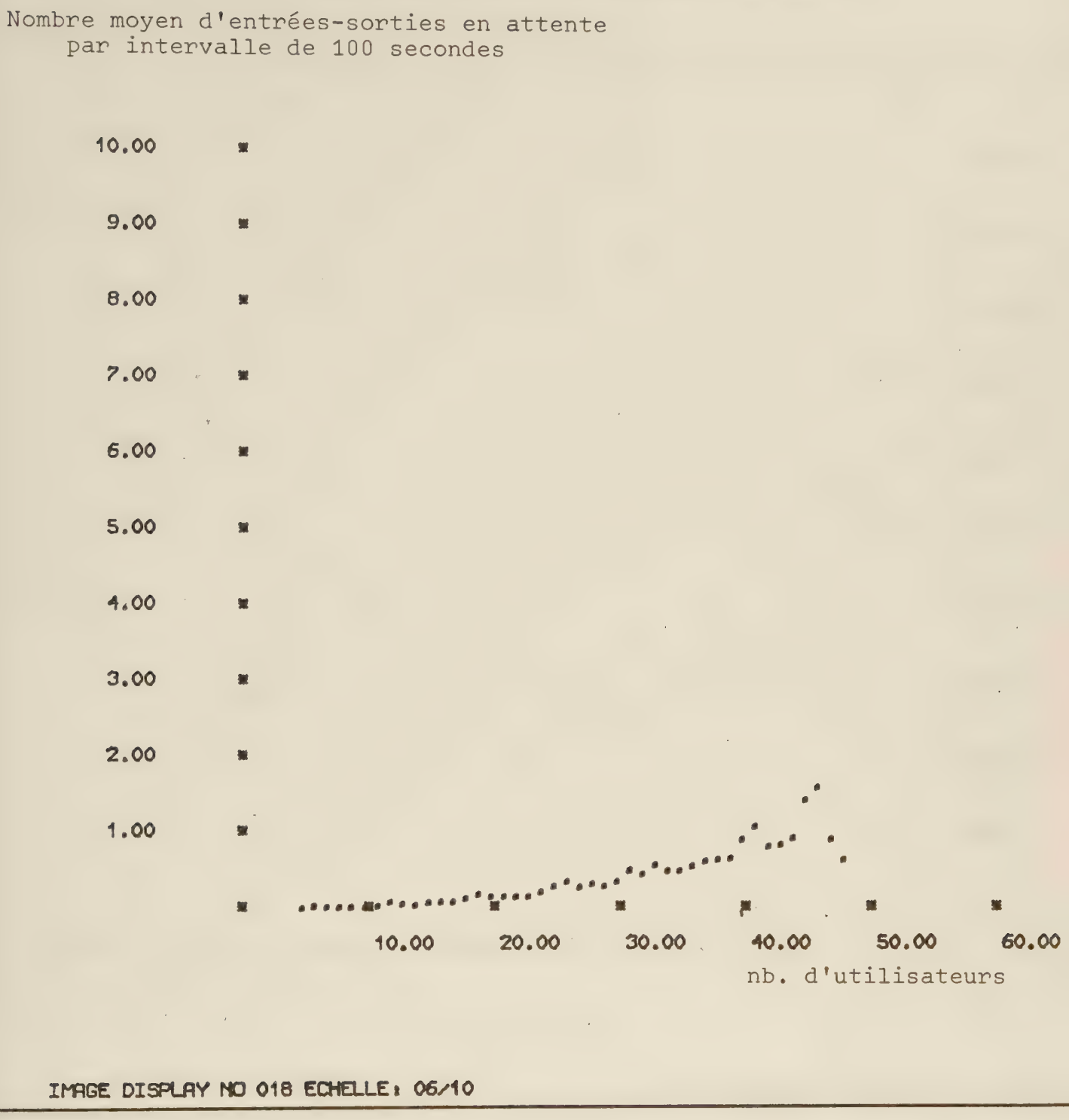


Figure 2.4 Queue disque avec S-ESPACE

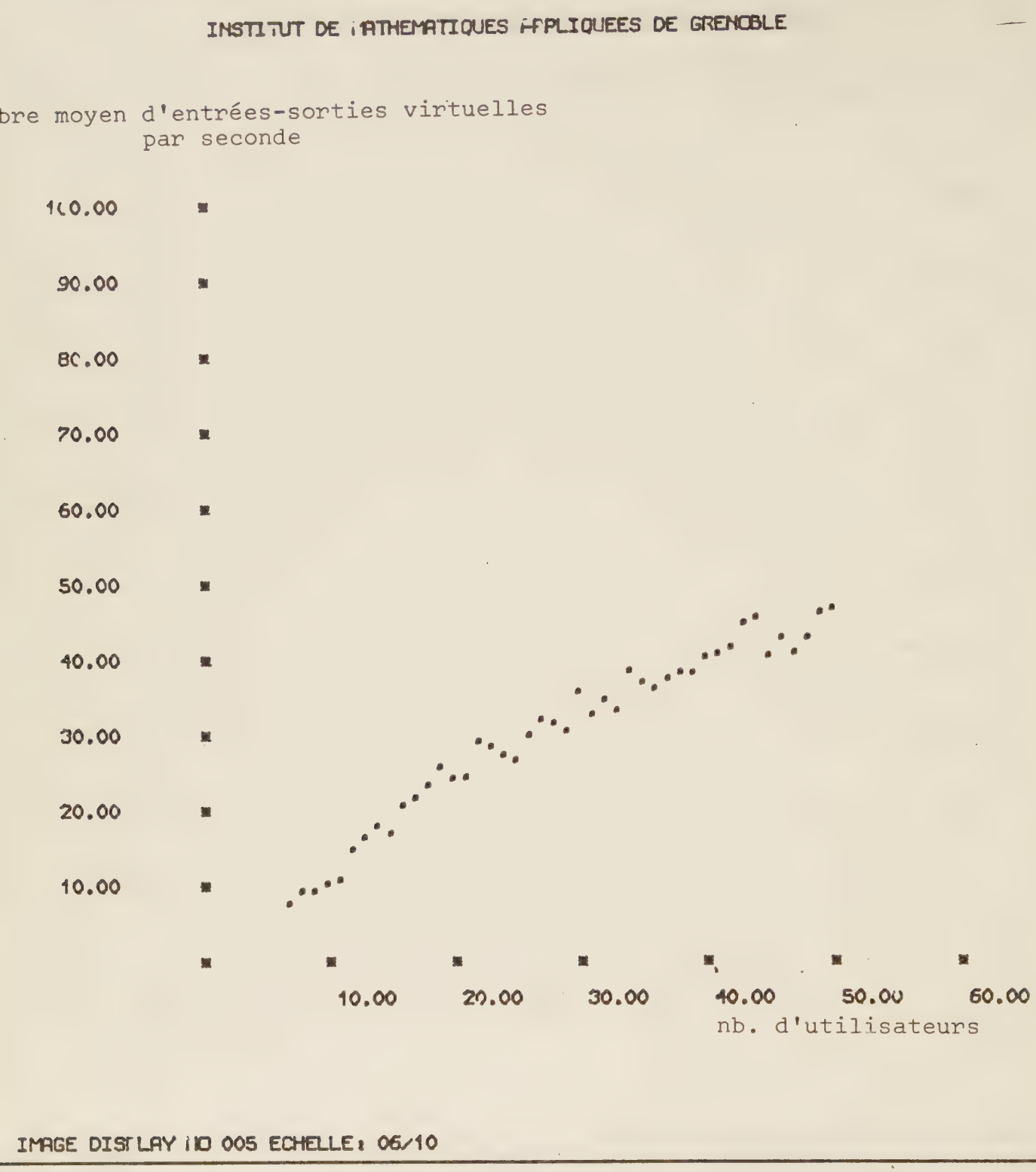


Figure 2.5 E/S virtuelles sans S-ESPACE

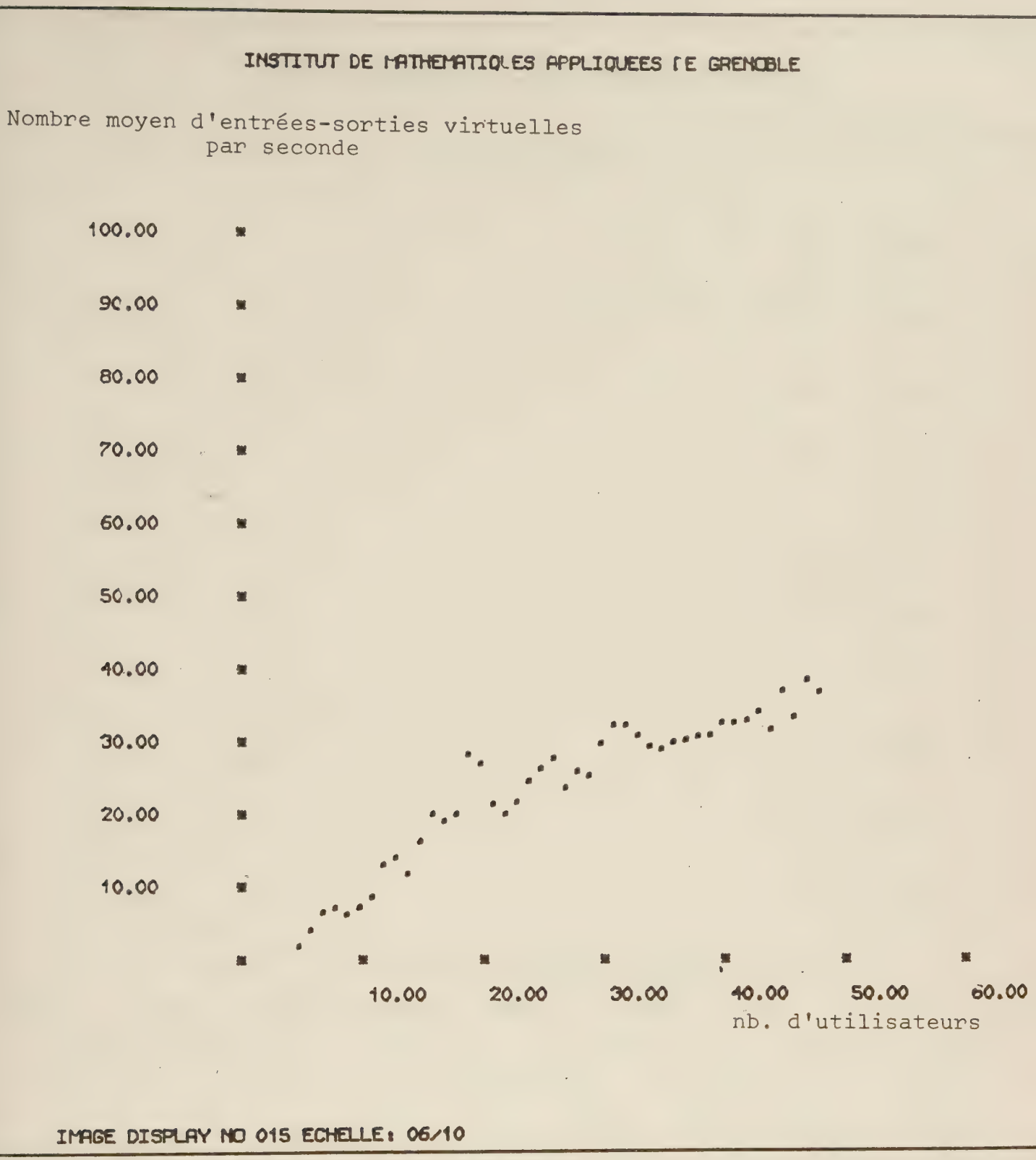


Figure 2.6 E/S virtuelles avec S-ESPACE

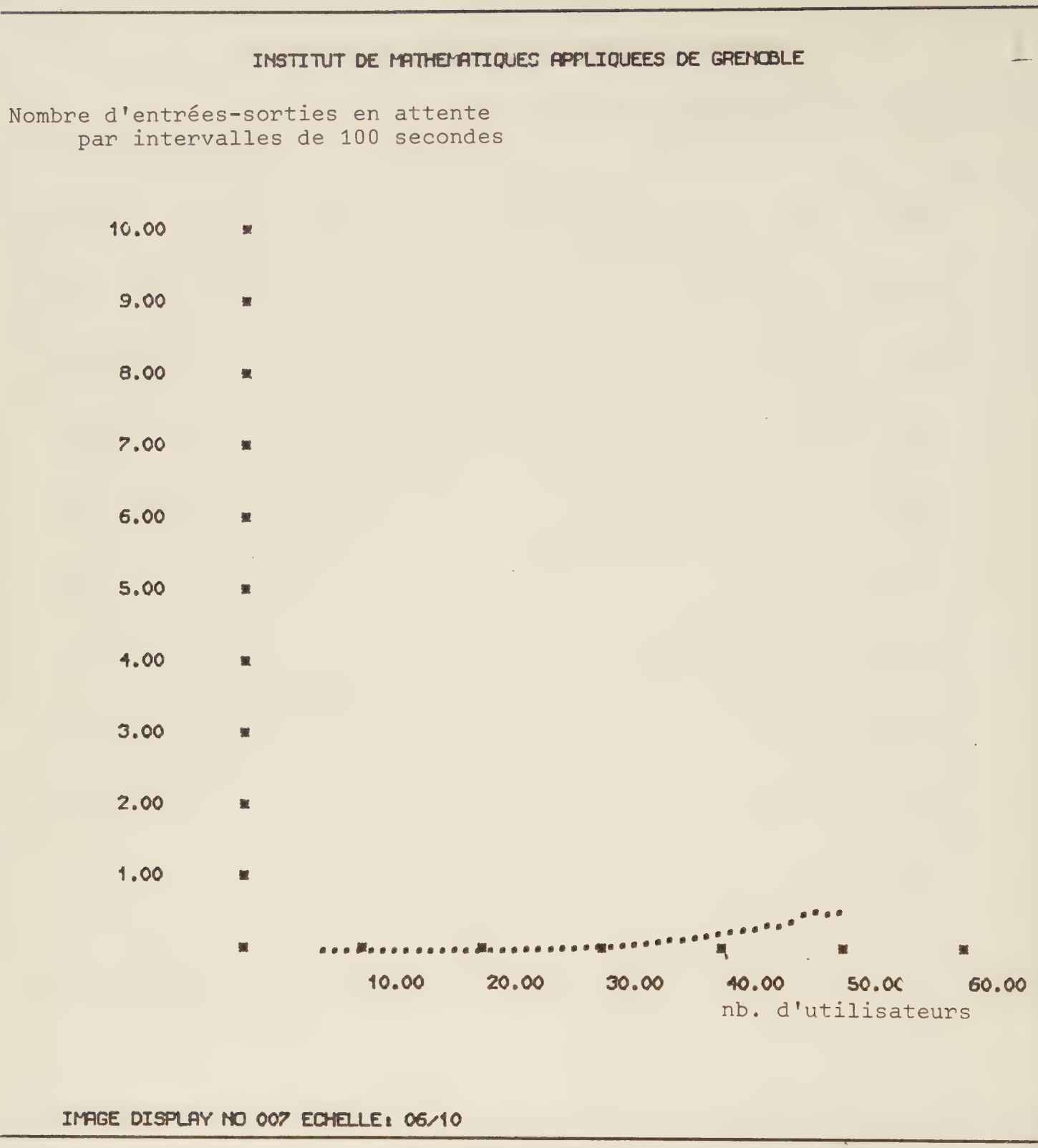


Figure 2.7 Queue tambour sans S-ESPACE

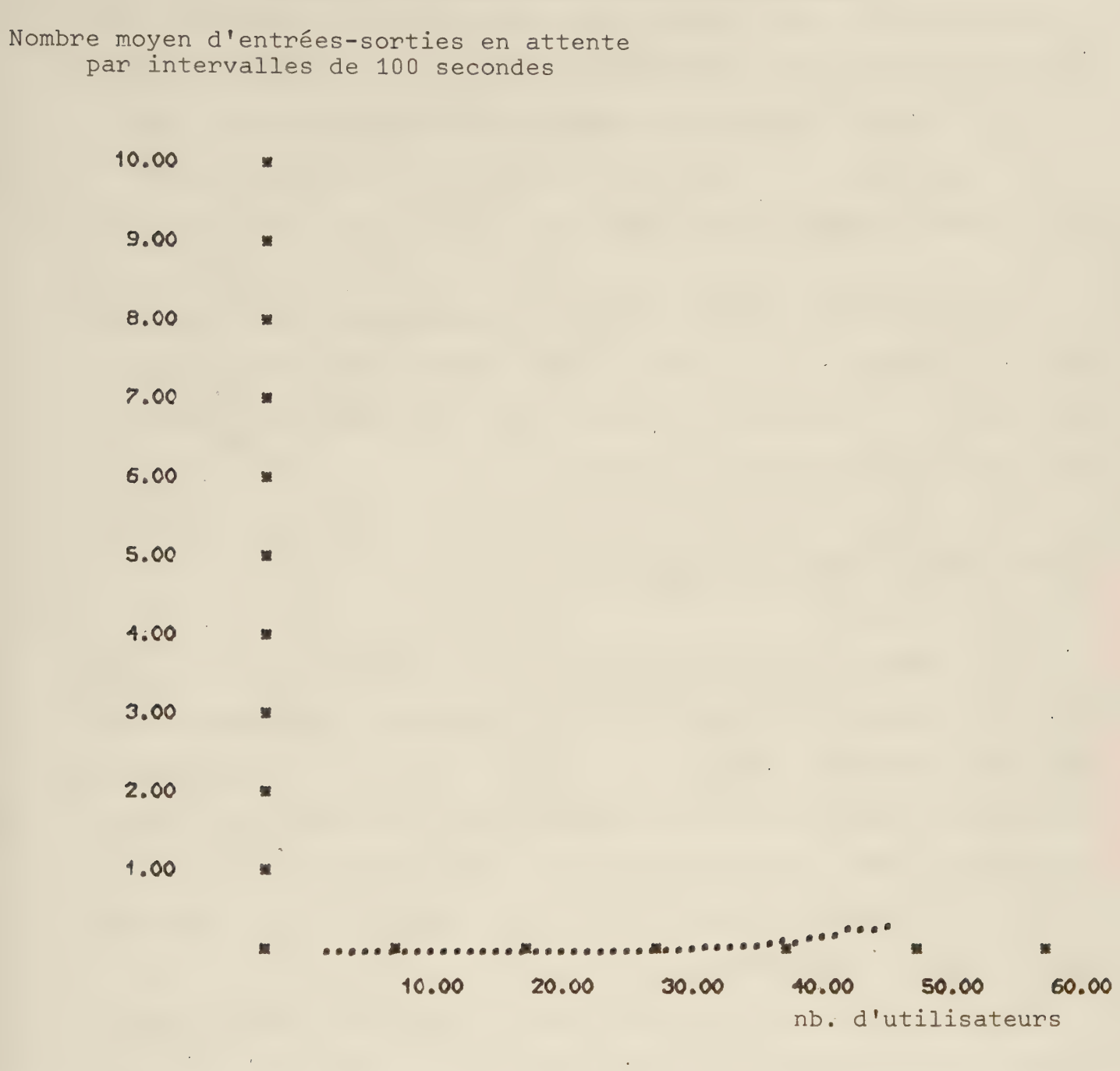


Figure 2.8 Queue tambour avec S-ESPACE

5.3.4) Pagination.

Les courbes des figures 2.9, 2.10 représentent le nombre d'opérations de pagination éxécutées par CP. On constate là aussi une baisse ce qui confirme nos suppositions précédentes.

5.3.5) Temps en mode superviseur.

Les courbes des figures 2.11, 2.12 représentent le temps passé dans le système CP. Le fait d'avoir diminué les entrées-sorties virtuelles et la pagination a contribué à diminuer son travail ce qu'on peut constater sur les courbes.

5.3.6) Temps en mode problème.

Les courbes des figures 2.13, 2.14 montrent que le temps problème a baissé en dessous de 30 utilisateurs. Ceci vient du fait que les machines virtuelles ont eu moins d'entrées-sorties virtuelles à éxécuter et que le travail des utilisateurs est resté à peu près le même. On constate que le temps passé par le système à attendre a augmenté, ce qui montre que le système n'est pas chargé avec ce nombre d'utilisateurs.

Par contre au-dessus de 30 utilisateurs le temps problème devient plus important quand on utilise le S-ESPACE. On a donc repoussé légèrement le seuil de saturation du système.

5.4) Conclusion.

La première conclusion que nous pouvons tirer de ces mesures est que nous n'avons pas dégradé les performances globales du système. D'autre part les modifications faites à CMS sont mineures.

Dans la gestion de fichiers de CMS+ ces mécanismes seront employés pour tous les fichiers. Nous espérons avoir des performances acceptables.

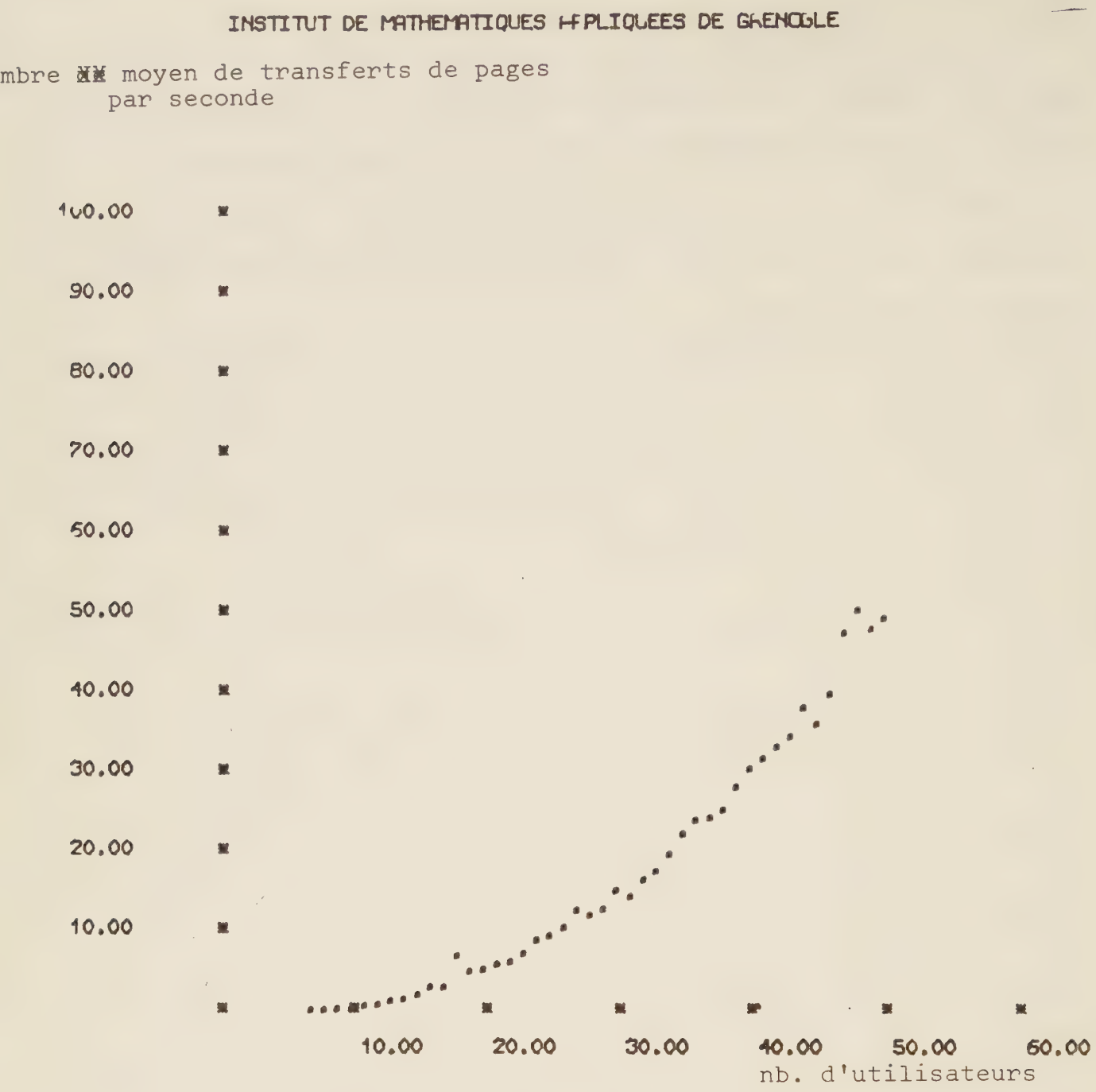


Figure 2.9 Pagination (interruptions pour pages manquantes)
sans S-ESPACE

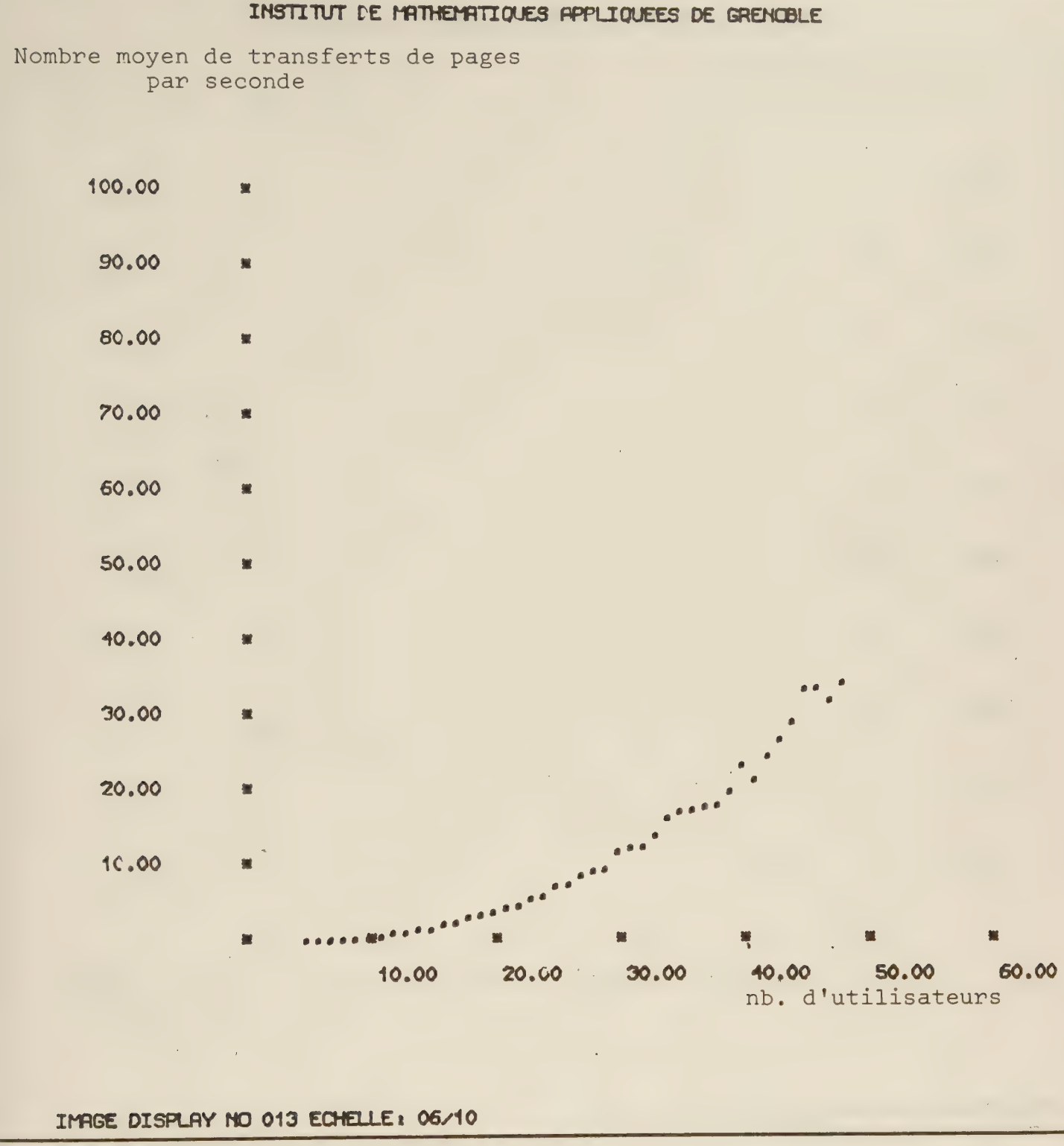


Figure 2.10 Pagination avec S-ESPACE

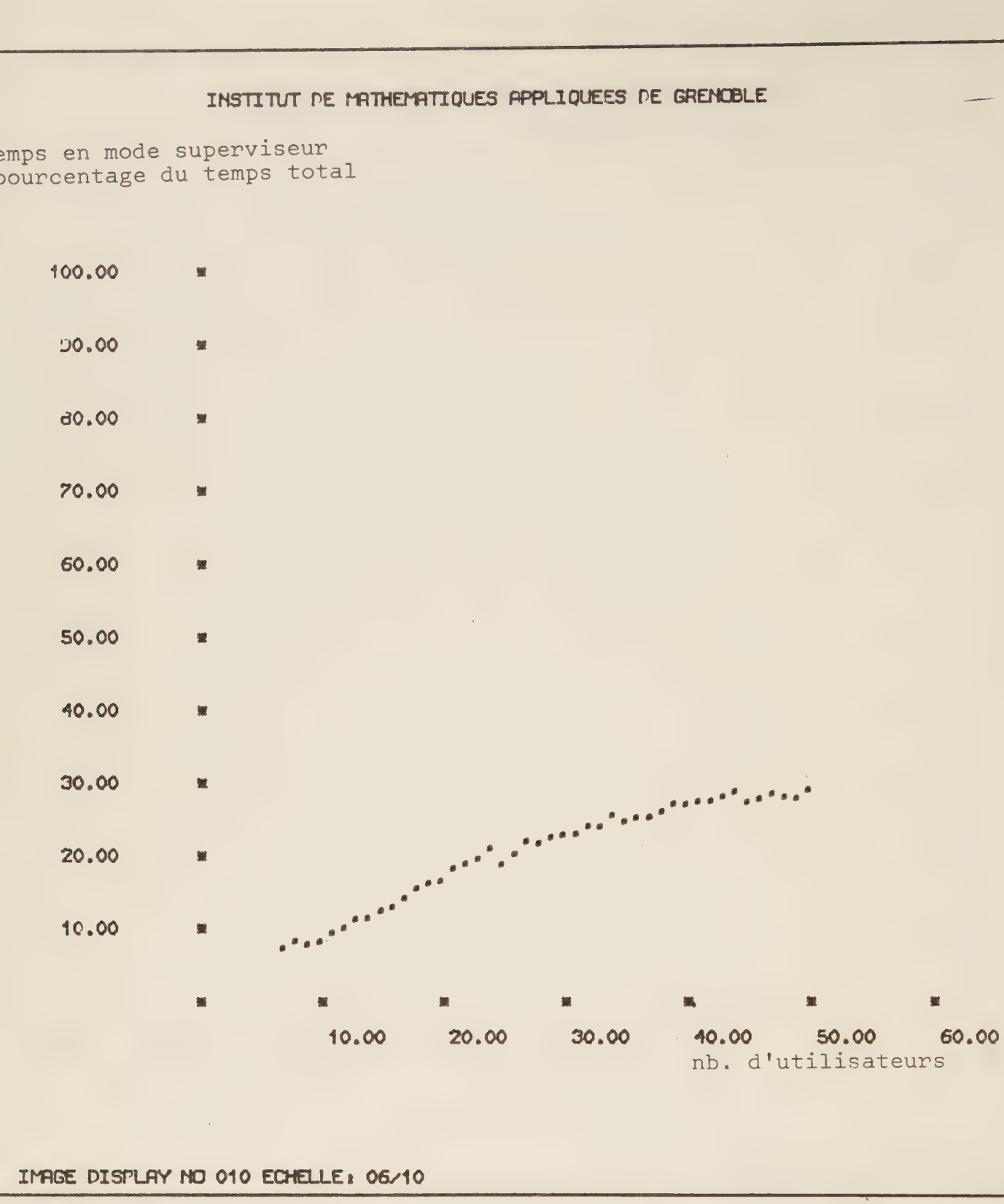


Figure 2.11 Temps CP sans S-ESPACE

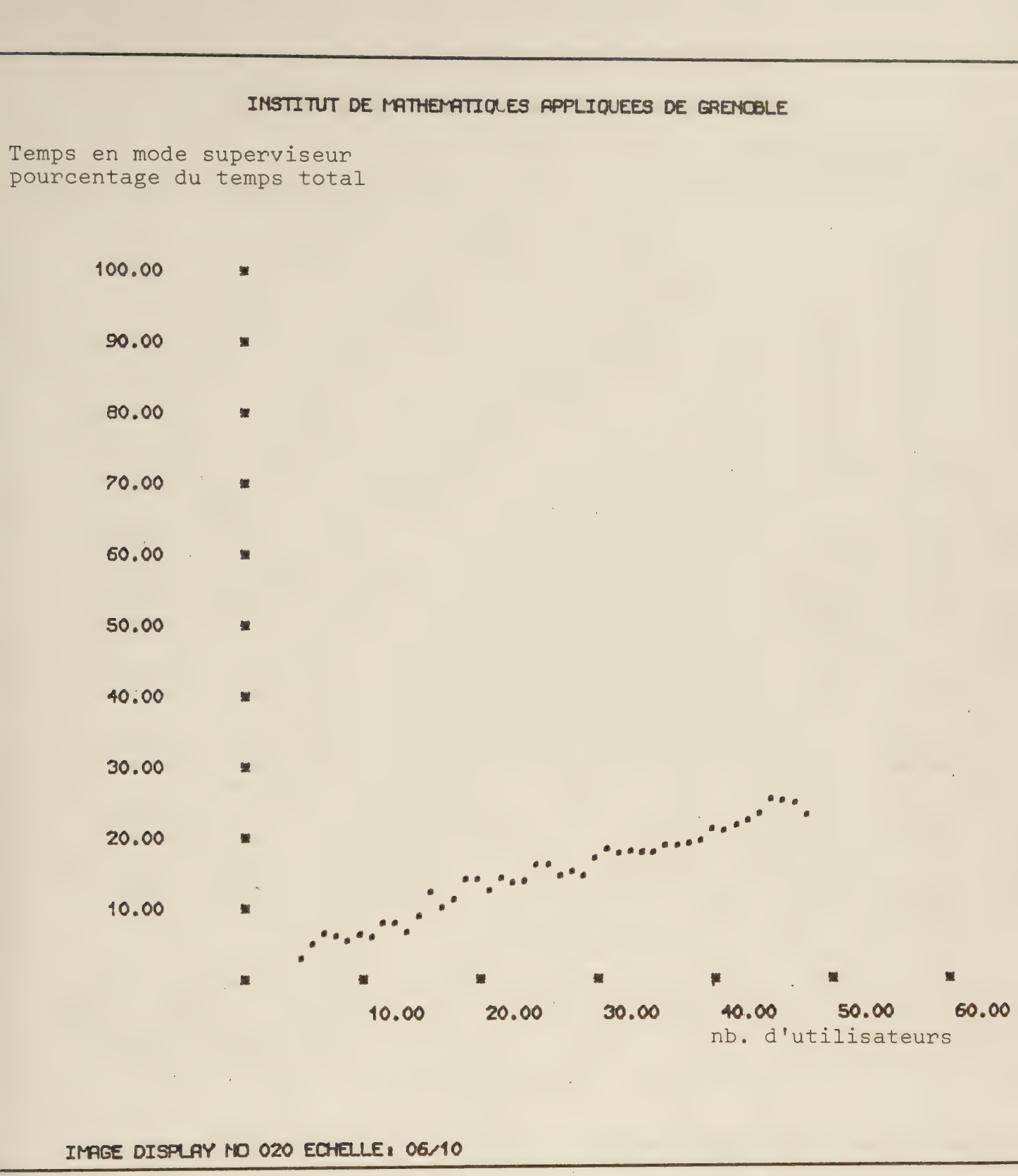


Figure 2.12 Temps CP avec S-ESPACE

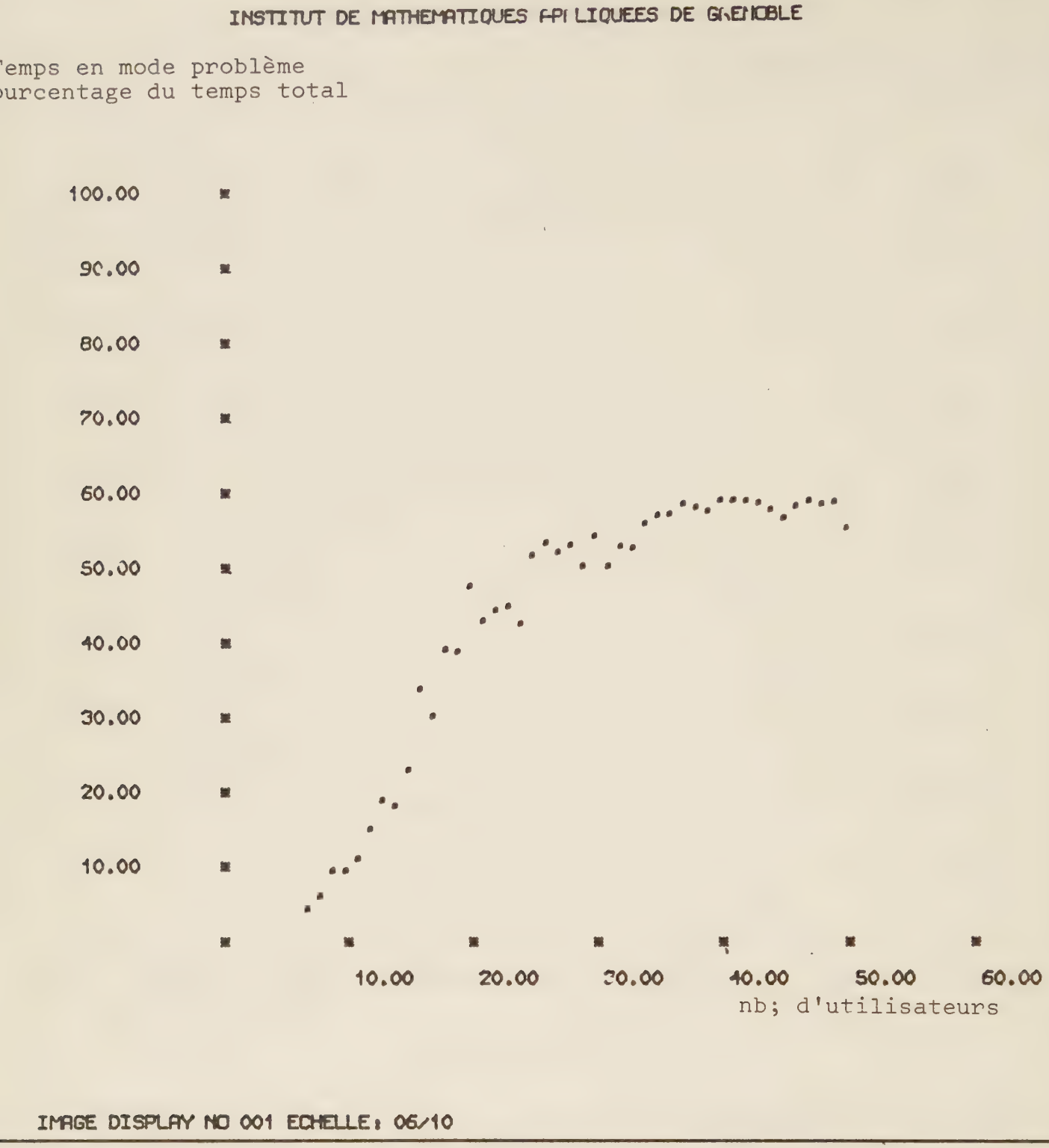


Figure 2.13 Temps problème sans S-ESPACE

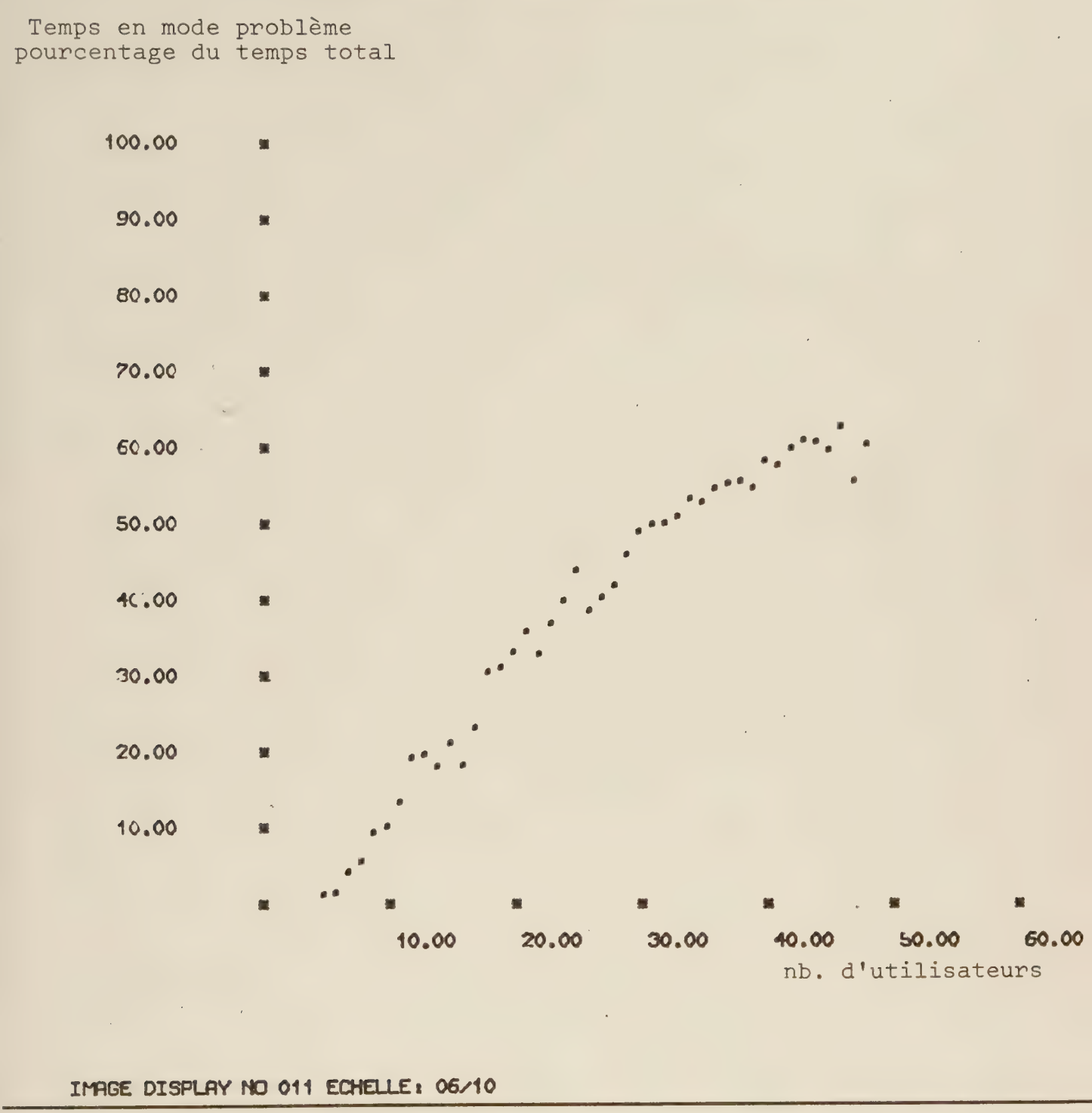


Figure 2.14 Temps problème avec S-ESPACE

C H A P I T R E 3

M - E S P A C E S

Comme on l'a vu au chapitre précédent, l'idée d'exploiter les possibilités de CP depuis une machine virtuelle (appelée alors environnement) a permis de définir des unités périphériques homogènes structurées en pages. De la même manière on peut envisager d'utiliser CP pour modifier en cours de fonctionnement un environnement et en particulier sa mémoire afin d'obtenir une gestion efficace des ces unités. Dans cette partie on montre la réalisation de la variation dynamique de la taille de la mémoire d'un environnement, l'utilisation de la segmentation du 360/67 pour effectuer le partage et la protection de la mémoire.

1) DEFINITION.

Un M-ESPACE est constitué d'une suite de segments Sj ($0 \leq j \leq Smax$), chaque segment étant lui-même constitué d'une suite de pages Pj ($0 \leq j \leq Pmax$). L'adresse d'une page dans cet espace est le couple (numéro de segment, numéro de page).

On remarque qu'il peut y avoir une discontinuité entre la plus

haute adresse du segment Sj et la première adresse du segment Sj+1. C'est une des propriétés fondamentales des M-ESPACE.

Dans le cas de l'IBM 360/67 on a 16 segments de 1024k soit:

Smax=15 et Pmax=255

Dans le cas des IBM 370 on peut avoir :

- si les segments sont de 1024k

  Smax=15 et Pmax=511 (pages de 2k)

  Smax=15 et Pmax=255 (pages de 4k)

- si les segments sont de 64k

  Smax=255 et Pmax=31 (pages de 2k)

  Smax=255 et Pmax=15 (pages de 4k)

2) CARACTERISTIQUES.

2.1) Partage de la mémoire.

Les systèmes paginés partagent la mémoire au moyen des tables décrivant la mémoire virtuelle. Par exemple si deux tables de pages ont une entrée décrivant la même page réelle, cette page est commune aux deux espaces virtuels. Sur le 360/67 où une mémoire virtuelle est décrite par des tables à deux niveaux: tables de segments, tables de pages, le partage est donc possible au niveau du segment et de la page.

Considérons sous CP N utilisateurs qui se partagent P pages ne constituant pas un segment (cf fig. 3.1). Les P pages sont décrites N fois par les N tables de pages et de pages externes des machines virtuelles. La recopie dans l'espace de pagination d'une des P pages nous impose de connaitre la liste des utilisateurs de cette page afin de mettre à jour les N tables décrivant cette page. A l'inverse, lorsqu'il se produit intérruption pour page manquante, la

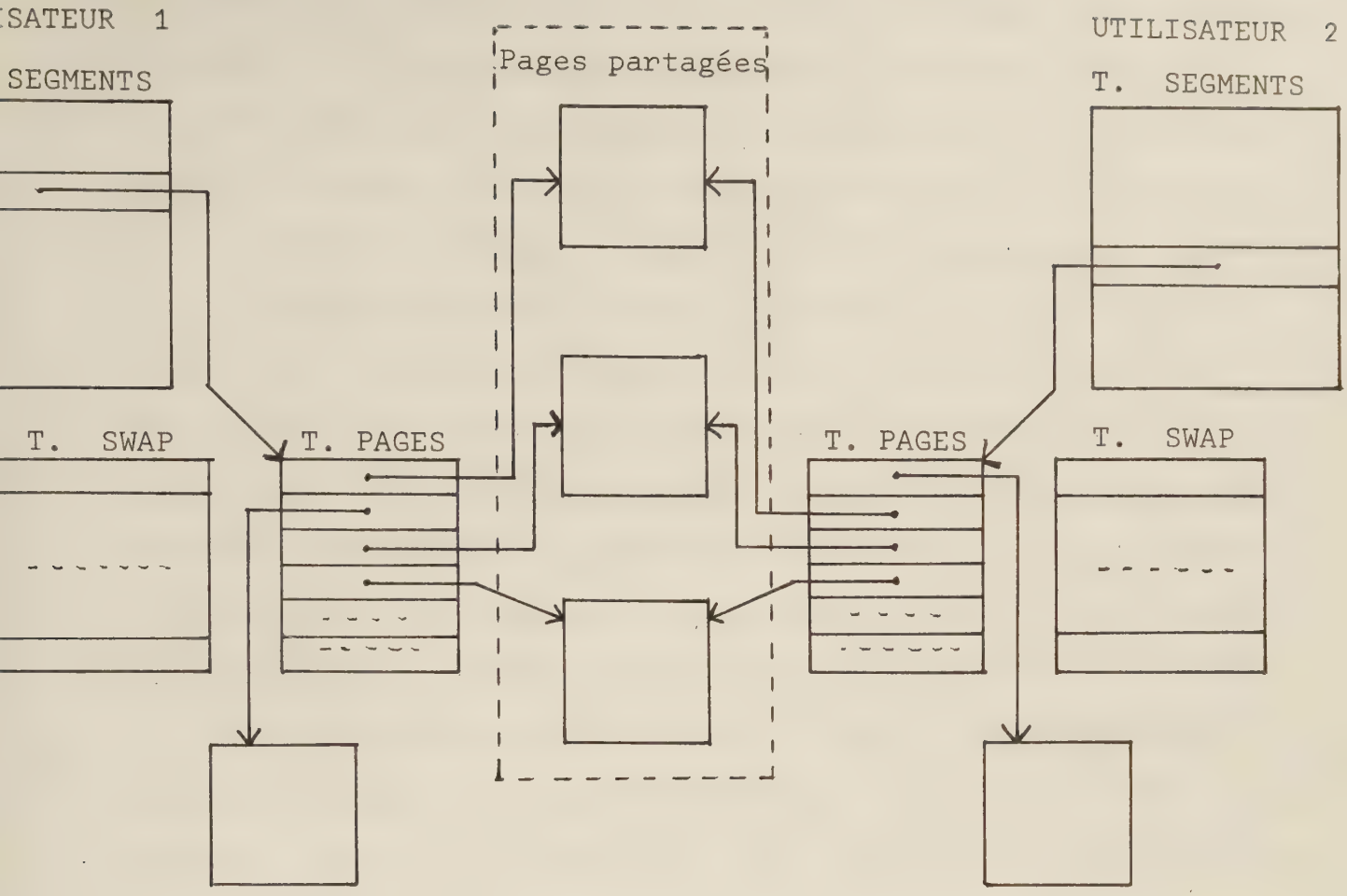


Figure 3.1 Partage de p pages

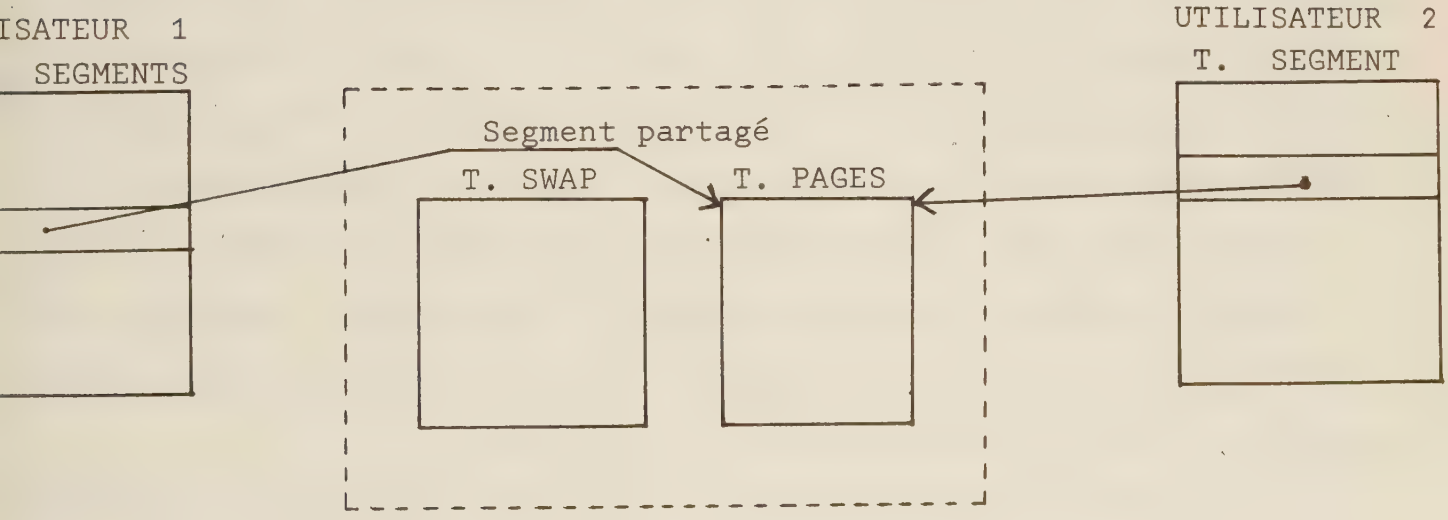


Figure 3.2 Partage d'un segment

recopie de la page en mémoire réelle nous impose a nouveau de mettre à jour les N tables. Toute opération de pagination demande donc un parcours de la liste des utilisateurs attachée à une page. Un tel mécanisme n'est pas prévu dans CP. La solution actuelle (partage des pages réentrantes de CMS) consiste à bloquer en mémoire les pages partagées de sorte qu'elles ne soient jamais déplacées.

Par contre si les N machines virtuelles se partagent un segment complet (cf fig. 3.2), il est décrit par une seule table de pages et une seule table de pages externes repérées par les N tables de segments des utilisateurs. Lorsque CP éxécute une opération de pagination, il met à jour les deux tables, et toutes les mémoires virtuelles utilisant ce segment sont à jour. Le partage au niveau du segment peut donc se faire sans modifier la pagination de CP.

Cette notion est fondamentale et elle justifie le fait d'exploiter les propriétés de la segmentation.

2.2) Exploitation de la segmentation.

Le 360/67 (ou les 370) peut générer un espace d'adressage de 16 mégaoctets. Contrairement à certains systèmes (OS/VS1 par exemple) qui partagent l'ensemble de cet espace entre les utilisateurs, CP alloue à chaque environnement un espace d'adressage. Cette technique permet une protection mémoire totale entre les utilisateurs sans utiliser les clés de protection de la mémoire physique. Chaque utilisateur dispose d'un espace privé et il lui est impossible de générer une adresse qui référence la mémoire d'un autre (sauf évidemment en cas de partage).

On constate cependant que CP n'accorde aucune signification particulière aux segments, et les machines virtuelles disposent d'un espace continu d'adresse. Ceci vient du fait que la notion de

segments du 360/67 a été introduite essentiellement pour faciliter la gestion en mémoire réelle des tables qui décrivent les mémoires virtuelles. En effet, sans segmentation, une mémoire virtuelle de 16 mégaoctets serait décrite à l'aide d'une table de pages unique de 4096 entrées (8K octets), alors qu'avec segmentation elle est décrite a l'aide de 16 tables de pages de 256 entrées chacune. Il en résulte une gestion plus simple. D'autre part cette segmentation est transparente à l'utilisateur car elle ne se répercute pas sur l'adressage: la plus haute adresse du segment n est suivie de la première adresse du segment n+1. Cette segmentation est donc fondamentalement différente de celle du GE 645 par exemple où chaque segment constitue un espace d'adressage (01, D3).

On se propose d'utiliser ces mécanismes de segmentation pour créer des segments logiques. Par définition on appelle segments logiques des entités d'informations indépendantes les unes des autres qui résident en mémoire virtuelle. Il y a en fait deux moyens de créer de telles entités: découper la mémoire virtuelle en sous-ensembles et les gérer par software ce qui est coûteux, ou établir une bijection entre les segments logiques et les segments virtuels du 360/67. Cette deuxième solution qui a été employée présente l'avantage d'utiliser le hardware pour résoudre les problèmes d'accès et de protection (cf 2.4). En contre-partie il y a deux inconvénients: on a au plus 16 segments logiques, et les segments virtuels sont très grands (1024K octets) et on aura rarement des segments logiques de cette taille. Le premier nous imposera donc de minimiser le nombre des entités logiques (on ne pourra pas associer a un segment logique un fichier comme MULTICS). Par contre on peut assouplir le deuxième inconvénient en exploitant la taille des segments qui réside dans la table des segments. Il a donc été possible de créer des segments logiques de n'importe quelle

taille (inférieure à 1024K). Cette segmentation se répercute sur l'adressage qui peut être discontinu. Chaque fois qu'un segment est incomplet, il ya une discontinuité avec le segment suivant.

2.3) Segment entité logique.

Le fait d'utiliser la segmentation permet de partitionner facilement la mémoire en ensembles disjoints et d'attribuer à chacun une fonction propre. Ceci évite une imbrication en mémoire entre les diverses parties du système et celles de l'utilisateur. Par exemple CMS structure sa mémoire de la façon suivante:

- noyau du système,
- zone de mémoire libre du système,
- zone utilisateur (à une adresse fixe),
- tables du chargeur.

En cours de fonctionnement, lorsque CMS n'a plus de mémoire libre dans sa zone, il en prend dans celle de l'utilisateur: ceci peut être une source de conflit (effacement de zone de mémoire ce qui peut être grave si c'est le catalogue des fichiers). D'autre part on constate que si l'accroissement du noyau de CMS (par adjonction de nouvelles fonctions) impose de déplacer l'adresse de la zone utilisateur, il faut regénérer toutes les commandes de type MODULE (cf chapitre 2 4.1) qui s'éxécutent dans cette zone.

CMS+ exploite la segmentation. On adopte le découpage suivant:

Segment 0 mémoire libre du noyau de CMS+

Segment 1 noyau réentrant de CMS+ (partageable)

Segment 2 mémoire libre de l'utilisateur

Segment 3-14 descripteurs de 1-ESPACES (partageables, cf chapitre 4)

Segment 15 réservé à un système de mise au point (SPY L2)

Ainsi chaque segment est considéré comme une entité logique. Chacun peut varier en taille (dynamiquement comme on le verra par la suite) sans influencer le comportement global du système et surtout l'adressage des autres parties.

2.4) Protection de la mémoire.

Les machines virtuelles ont à leur disposition des mécanismes classiques de protection de la mémoire: clés associées aux blocs de mémoire réelle. Dans une mémoire structurée en segments il peut être intéressant de protéger ces segments entre eux. En effet chaque segment représente une entité logique et les références inter-segment doivent être faites avec précaution. Par exemple si on reprend le découpage de CMS+ du paragraphe précédent, un programme utilisateur résidant dans un segment ne doit pas en référencer un autre. Il demande ce service au système au moyen de l'instruction SVC (appel au superviseur). Par contre le segment contenant le noyau de CMS+ est habilité à référencer tout segment, de même que les routines de la gestion de fichier peuvent référencer les segments de description des 1-ESPACE.

Le 360/67 (et les 370) ne dispose pas de mécanismes de protection de la mémoire par clé au niveau du segment. Mais on remarque que si on invalide une entrée de la table de segment on interdit tout accès à ce segment. On a donc par ce biais une protection en lecture et en écriture, c'est à dire une protection absolue d'un segment. Un environnement peut demander à CP+ (par l'instruction DIAGNOSE) d'invalider ou de rendre valide un segment en précisant son numéro.

Cette technique est donc équivalente à une protection de la mémoire où le bloc protégé est le segment, et où les clés ont deux

valeurs: accès autorisé ou interdit. Elle n'est valable que pour un seul utilisateur. Dans le cas d'un segment partagé elle est encore opérationnelle. En effet si N utilisateurs se partagent un segment chacun a une table de segments. Rendre un segment invalide pour l'un d'eux revient à invalider une entrée dans sa propre table de segments. Par contre le problème de protection d'un segment partagé entre plusieurs utilisateurs n'est pas traité ici.

2.5) Croissance dynamique des segments.

La définition d'un M-ESPACE, due à l'implémentation actuelle, se fait de façon statique dans le catalogue de CP+ qui décrit les machines virtuelles. En cours de fonctionnement il se peut qu'un utilisateur ait besoin de plus de place qu'il n'en dispose dans un segment. Une première solution statique consiste à modifier la configuration de l'environnement dans le catalogue. C'est seulement au prochain LOGIN que la modification sera prise en compte. Une autre solution consiste à demander à CP+ une extension de mémoire, dynamiquement en cours de fonctionnement sans perturber le système. Ceci revient à rallonger la table de pages d'un segment ou à créer un nouveau segment.

Il est bien évident que le système doit être adapté à cette croissance dynamique. Par exemple sous VM/370 (I10) (nouvelle version de CP pour IBM 370) il existe une commande qui permet aux utilisateurs de modifier leur taille mémoire à tout instant. Mais de par la constitution de CMS cette modification entraine un nouveau chargement de CMS pour qu'il tienne compte d'une éventuelle extension.

Dans le cas de CMS+ chaque segment est indépendant et peut donc varier en taille dynamiquement. Il est par exemple possible de faire

varier la taille des descripteurs de 1-ESPACE suivant le nombre de fichiers de l'utilisateur.

3) APPLICATION DANS L'ESPACE PHYSIQUE.

La mémoire de type M-ESPACE est décrite à l'aide du même mécanisme que la mémoire continue. Il existe pour chaque utilisateur une table de segments et à chaque entrée de cette table est associée: un couple table de pages, table de pages externes si le segment est valide; aucune table s'il est invalide. La différence avec la mémoire continue est que tout segment peut être incomplet.

Jusqu'à présent lorsqu'un environnement référence une adresse deux cas se présentent:

a) Le hardware effectue la traduction en une adresse réelle.

b) Il y a interruption pour page manquante et on a deux possibilités:

- l'adresse virtuelle est invalide car elle est en dehors de l'espace d'adressage alloué à l'utilisateur (il est possible d'allouer des espaces d'adressage entre 8K et 16 mégaoctets).
- l'adresse virtuelle est valide et il faut amener la page en mémoire par une opération de pagination.

Dans le cas de la mémoire discontinue le cas b) se complique car une adresse peut être invalide dans deux cas: d'une part si elle est en dehors de l'espace d'adressage de l'utilisateur comme ci-dessus, et d'autre part si elle référence la partie non allouée entre deux segments (cas d'une discontinuité). Il faut donc avoir un test plus sélectif et vérifier dans la table de segments que le numéro de la page demandée se trouve effectivement dans la partie allouée du segment. Si c'est le cas on initialise le transfert de la page vers la mémoire réelle, sinon une interruption pour adressage invalide est reflétée à l'environnement.

Le problème est plus simple à résoudre sur 370 grâce à

l'instruction LRA (load real address) qui opère comme le test sélectif. Une instruction suffit, alors qu'il en faut une dizaine sur le 360/67.

4) REALISATION SOUS CP.

4.1) Définition dans le catalogue.

Dans la version de base de CP, la mémoire est définie dans le catalogue qui décrit les configurations des machines virtuelles à l'aide du mot-clé: 'CORE'.

Ex:

```
CORE    256K
```

Dans CP+ nous avons étendu cette définition pour définir les M-ESPACE. On décrit successivement la taille de chaque segment.

Ex:

```
CORE    256K,48K,,512K
```

Ceci signifie:

```
Segment  0  256K
         1   48K
         2  invalide (vide)
         3  512K
         4-15 invalides (implicitement)
```

Par convention deux virgules successives indiquent un segment vide et si la définition ne tient pas sur une ligne elle peut se poursuivre sur plusieurs.

Ex:

```
CORE   256K,128K,,512K,768K,
       1024K,,1024K,48K
```

4.2) Activation des M-ESPACE par CP.

Lorsqu'un utilisateur entre dans le système à l'aide de la

commande LOGIN, CP lit dans le catalogue la taille des différents segments et construit autant de tables de pages et pages externes qu'il est nécessaire.

En cours de fonctionnement CP+ réfléchit aux environnements les interruptions pour adressage invalide si elles résultent d'une référence à une discontinuité.

Certaines commandes ont été modifiées pour faciliter le travail de l'utilisateur. Ainsi:

DISPLAY ADD

précise à l'utilisateur en cas d'erreur si l'adresse indiquée est en dehors de son espace d'adressage, ou s'il s'agit d'une discontinuité. De même la commande DUMP peut imprimer le contenu de la mémoire utilisateur sous forme héxadécimale en sautant les discontinuités.

5) EXTENSION DES MECANISMES.

5.1) Croissance dynamique des segments.

La réalisation actuelle des M-ESPACE sous CP est statique. C'est l'admnistrateur du système qui est habilité à modifier le catalogue de description des machines pour changer la taille des segments.

Dans CP+ par contre, où CMS+ associe à chaque segment un rôle particulier, il est intéressant de les faire varier dynamiquement en taille. Il suffit de disposer de deux primitives invoquant CP+:

AJOUTER (N,S)

ENLEVER (N,S)

avec: N nombre de pages

S numéro de segment

qui signifient d'ajouter ou d'enlever N pages à la fin du segment S.

A l'éxécution CP+ calcule la nouvelle taille du segment S et acquiert en mémoire libre réelle une nouvelle table de pages et de pages externes. Il y recopie les anciennes en les tronquant ou en les complétant suivant le cas. Il libère ensuite les anciennes tables. CP+ renvoie éventuellement un code d'erreur si le segment dépasse une taille maximum autorisée (dans le catalogue par exemple) ou 1024K.

Ces fonctions permettent aussi d'ajouter des segments à un environnement. En effet si le segment S n'existe pas lorsque CP+ reçoit la primitive AJOUTER, il crée un nouveau segment. De même lorsqu'on enlève toutes les pages d'un segment CP+ le détruit et il devient invalide.

Dans le cas d'un segment partagé, les systèmes virtuels qui l'utilisent doivent être capables de s'apercevoir de la modification

de taille de celui-ci. Par exemple on peut adopter la solution de placer la description du segment dans le segment lui-même. Lorsqu'un des systèmes change la taille, il met à jour la description et les autres utilisent directement la nouvelle taille.

5.2) Protection au niveau du segment.

Comme on l'a vu le fait d'invalider une entrée de la table de segments en interdit tout accès. C'est donc une protection absolue du segment. On rappelle que cette technique n'est opérationnelle que pour l'environnement d'un utilisateur (cf 2.4).

Cette protection peut être réalisée à l'aide de trois primitives invoquant CP+ (à l'aide de l'instruction DIAGNOSE):

BLOQUER (S) rendre invalide le segment numéro S

DEBLOQUER (S) rendre valide le segment S

BLOQUERSAUT (S,ADD) rendre invalide le segment S et se brancher à l'adresse ADD n'appartenant pas à S

Au niveau de CP+ bloquer ou débloquer un segment revient à mettre ou enlever le bit 'segment invalide' dans l'entrée numéro S de la table de segment de l'utilisateur.

La troisième primitive est nécessaire si un programme veut bloquer le segment dans lequel il se déroule, et donner le contrôle à un programme dans un autre segment. C'est le cas par exemple de CMS+ qui active un programme utilisateur (cf 2.3). En effet si on emet:

```
BLOQUER (S)
BRANCH  ADD
```

l'opération de blocage se déroule normalement mais lorsque l'instruction de branchement s'éxécute, le segment est déjà bloqué et il y a interruption pour adresse invalide. Par contre si on

utilise BLOQUERSAUT c'est au niveau de CP+ que se fait le branchement. En fin du programme utilisateur le retour à CMS+ peut se faire soit par une instruction SVC, soit en se branchant à une adresse particulière du noyau.

Lorsque CP+ reçoit une interruption qu'il doit réfléchir à un environnement qui a invalidé des segments (entrée-sorties, SVC, programme, etc...) il doit s'assurer que la routine de traitement n'est pas dans un segment invalide. Une première solution consiste à débloquer tous les segments. Une deuxième solution plus sélective consiste a débloquer le segment 0 pour accéder au mot d'état de la machine (PSW) correspondant à l'interruption. Il suffit ensuite d'examiner l'adresse contenue dans le mot d'état et de débloquer le segment qu'elle référence. La routine de traitement peut être activée, et comme c'est en général une routine du système virtuel, elle peut à son tour débloquer d'autres segments.

Lorsque CP+ reçoit une interruption indiquant qu'un segment virtuel est invalide, il réfléchit à l'environnement une interruption pour violation de protection mémoire, en suivant la procédure décrite ci-dessus.

Enfin si les primitives désignent des segments en dehors de l'espace d'adressage de l'utilisateur CP+ renvoie un code d'erreur.

5.3) Partage des segments.

Lorsqu'un utilisateur entre dans le système, CP+ lui construit un environnement muni d'une mémoire complète sans aucun segment partagé. Si cet utilisateur désire partager un segment il doit le demander à CP+ car le partage ne sera effectif que lorsque les tables le décrivant seront en commun (cf 2.1).

Remarquons que le partage impose un protocole d'accord entre les

utilisateurs. En effet les systèmes utilisés doivent être capables de conserver la cohérence de l'information partagée. D'autre part il doit exister un certain nombre de protections pour éviter que n'importe qui puisse partager n'importe quoi (disparition de la protection mémoire entre les utilisateurs). Il est donc préférable d'étudier les cas ou le partage est souhaitable et de le faire éxécuter automatiquement par CP+, plutot que de fournir des primitives aux utilisateurs ainsi qu'un système de mots de passe qui entrainerait des conflits.

On munit CP+ de deux fonctions non accessibles aux utilisateurs:

LIER (S1,USER1,S2,USER2)

qui signifie: lier le segment S2 de l'utilisateur USER2 comme segment S1 de l'utilisateur USER1;

DELIER (S1,USER1)

qui signifie: reconstruire a l'utilisateur USER1 un segment S1 vierge.

L'éxécution de la fonction LIER consiste à libérer les tables décrivant le segment S1 de USER1 et de placer dans l'entrée S1 de sa table de segment l'entrée S2 de celle de USER2.

L'éxécution de la fonction DELIER consiste à reconstruire une table de pages et de pages externes dont l'adresse est placée dans l'entrée S1 de la table de segment de USER1.

Remarquons que la fonction DELIER est éxécutée chaque fois qu'un utilisateur abandonne le travail en cours et en initialise un autre (par exemple à l'aide de la commande IPL).

Prenons comme exemple de partage le noyau de CMS+ (placé dans le segment numéro 1 de tous les utilisateurs cf 2.3). Lorsqu'un utilisateur initialise le chargement de CMS+, CP+ qui connait sa structure sait déjà que l'utilisateur désire partager le segment contenant le noyau. Il traite de façon spéciale le premier

utilisateur (PREMIER). Il lit effectivement le noyau de CMS+ dans le segment numéro 1 de PREMIER. Pour les utilisateurs suivant il se contente d'éxécuter:

LIER (1,USER,1,PREMIER)

ce qui donne l'accès au noyau de CMS+ sans faire de chargement en mémoire.

Un autre cas possible de partage est celui des descripteurs des 1-ESPACE qui sera étudié au chapitre 4.

C H A P I T R E 4

2 - E S P A C E S

Après avoir défini des unités logiques (1-ESPACE) on a réalisé une gestion de fichiers les utilisant. On se trouvait confronté à deux problèmes : partager un 1-ESPACE en sous-ensembles disjoints constituant des fichiers d'une part, et définir des moyens d'accès simples assurant une bonne protection entre les fichiers. On a résolu les deux problèmes en donnant aux fichiers la structure d'une mémoire virtuelle.

1) CONCEPT DE 2-ESPACE.

1.1) Définition.

Un 2-ESPACE est par définition une suite ordonnée d'octets numérotés de 0 à 2**31. Cette suite réside effectivement dans des pages non nécessairement contiguës d'un 1-ESPACE.

Le fichier dans le système CMS+ est un 2-ESPACE. Un 1-ESPACE comprend donc une suite de 2-ESPACE ainsi qu'un catalogue qui permet de distinguer les 2-ESPACE entre eux.

1.2) Caractéristiques.

1.2.1) Protection.

Quelle que soit la manière de définir un 2-ESPACE, celui-ci est constitué d'un sous-ensemble de pages appartenant à un 1-ESPACE:

(a1,a2,a3,.....an) ai∈1-ESPACE

Par contre il y a plusieurs manières d'accéder aux éléments d'un 2-ESPACE. Une première méthode consiste à fournir un doublet:

(numéro de page dans le 1-ESPACE, déplacement dans la page)

Il est alors possible de lire la page demandée et d'accéder à l'information. Remarquons que cette méthode ne permet pas une protection efficace du 2-ESPACE. En effet on connait la liste des pages composant un 2-ESPACE. Mais lorsque l'utilisateur donne un numéro de page dans le 1-ESPACE on ne sait pas si la page référencée appartient au 2-ESPACE qu'il utilise. Dans ce cas la protection du 2-ESPACE impose à chaque accès de balayer la suite des pages le constituant pour s'assurer que la page référencée lui appartient. Cette méthode est peu économique en temps machine. D'autre part la gestion du 2-ESPACE en tant qu'espace d'adressage est à la charge de l'utilisateur.

La méthode que nous avons employé consiste à décrire les pages constituant le 2-ESPACE à l'aide d'une table de pages externes (analogue aux mémoires virtuelles) et de référencer les éléments par une adresse de 31 bits (1 mot soit 32 bits moins le bit de signe). C'est alors à la charge du système de déterminer où réside l'information demandée. En employant un système de découpage de l'adresse en groupes de bits qui servent à indexer la table de pages

externes, on obtient un accès rapide et une protection absolue. En effet lorsqu'on travaille sur un 2-ESPACE il est impossible de générer une adresse en référençant un autre. On retrouve une protection analogue à celle des 1-ESPACE, des mini-disques, des mémoires virtuelles.

1.2.2) 2-ESPACE et mémoire virtuelle.

Le 2-ESPACE se présente à l'utilisateur comme une mémoire virtuelle dont l'accès au nième octet se fait à l'aide de fonctions software. Ces fonctions (LIRE et ECRIRE cf 2.) effectuent les transferts de pages entre le 1-ESPACE et la mémoire virtuelle, et placent l'information dans la mémoire de l'utilisateur. Lorsqu'elles traitent une adresse d'un 2-ESPACE deux cas se présentent:

- la page du 2-ESPACE concernée est en mémoire virtuelle et on accède à l'information.
- la page concernée n'est pas en mémoire et on consulte la table de pages externes pour déterminer où elle réside dans le 1-ESPACE. Il suffit alors de la lire à l'aide des fonctions d'accès au 1-ESPACE.

Cette analogie permet à l'utilisateur de gérer le 2-ESPACE comme une mémoire. De plus dans la cas de CMS+ on a décidé de séparer entièrement la partie descriptive et le contenu des 2-ESPACE. En effet certains systèmes (TSS, OS/VSAM (I9)) considèrent aussi le fichier comme une mémoire découpée en blocs, mais certaines méthodes d'accès y placent des informations inconnues de l'utilisateur (clés). Celui-ci doit donc passer obligatoirement par les méthodes d'accès pour gérer le fichier, à moins qu'il ait connaissance du fonctionnement du système. En effet pour calculer une adresse dans le fichier il faut tenir compte de la taille de ces informations.

Dans le cas de CMS+ l'utilisateur pourra accéder au 2-ESPACE par les méthodes d'accès standard (définies ultérieurement) ou directement au caractère par les fonctions au niveau du 2-ESPACE. Il est libre de gérer cette 'mémoire' comme il le veut, et il peut même se définir de nouvelles méthodes d'accès si celles dont il dispose ne lui conviennent pas.

1.2.3) Allocation dynamique d'espace.

Lors de la création d'un 2-ESPACE aucune page du 1-ESPACE n'est reservée. L'utilisateur dispose d'un espace d'adressage potentiel de 2**31 octets. Toute lecture aura pour résultat une suite d'octets nuls car il ne contient rien.

Lorsque l'utilisateur écrit dans le 2-ESPACE de l'information deux cas se présentent:

- la page référencée n'existe pas
- la page existe

Dans le premier cas on alloue une page supplémentaire au 2-ESPACE et on y transfère l'information. Dans le second cas on se contente simplement d'effectuer le transfert.

On constate donc qu'il est possible d'écrire les enregistrements n'importe où dans cette espace, sans allouer l'espace physique intermédiaire.

Ex: écrire l'enregistrement numéro 10
écrire l'enregistrement numéro 100000
provoque l'allocation de deux pages seulement.

1.2.4) Simplicité d'emploi.

Le fait de considérer le 2-ESPACE comme un espace d'adressage

permet de rendre son emploi très simple. L'utilisateur nomme les éléments à l'aide d'une adresse et c'est le système qui se charge de faire le découpage en pages, et d'éxécuter les entrées-sorties.

1.2.5) Méthodes d'accès.

Les 2-ESPACE permettent de résoudre facilement l'accès aux enregistrements de longueur fixe de manière séquentielle ou directe en précisant le numéro de l'enregistrement. En effet accéder au nième enregistrement de taille T revient à calculer son adresse dans le 2-ESPACE:

$$AD=n*T$$

Les méthodes d'accès plus élaborées (variables, clés ..) ne sont pas encore complètement définies et seront traitées au niveau supérieur. Cependant dans cette première approche on a décidé de rester compatible avec la gestion de fichier de CMS. On a donc inclu des méthodes de traitement des enregistrements de taille variable en enfreignant la règle de séparer entièrement le 2-ESPACE et sa description. Les enregistrements de longueur variable ont le format suivant:

longueur,enregistrement,longueur, ...

Ceci permet de tester la réalisation sous CMS.

1.2.6) 2-ESPACE et fichier CMS.

Les fichiers de CMS+ bénéficient par rapport à ceux de CMS, des avantages des 1-ESPACE qui leur confèrent l'indépendance par rapport au support physique sur lequel ils résident, et l'utilisation des routines de pagination de CP+ pour les entrées-sorties. D'autre part ils bénéficient du concept d'espace d'adressage. Dans CMS+ on

fournit une adresse qui permet de référencer directement l'information grâce à la table de pages externes, alors que dans CMS il faut lire sur disque les adresses des blocs du fichier. On a donc un accès plus rapide avec CMS+.

Un autre avantage est la grande taille possible des fichiers de CMS+ (2**31 octets soit environ 20 disques de 100 millions de caractères), alors que les fichiers de CMS ont au maximum 13 millions de caractères.

1.3) Traduction des adresses.

Il s'agit de définir ici la fonction qui à partir d'une adresse A dans le 2-ESPACE détermine la page du 1-ESPACE où réside l'information:

$$f : A \longmapsto \begin{cases} (N,D) & N \text{ 1-ESPACE } 0 \text{ D } 4095 \\ \emptyset & \text{indéfini car la page n'est pas allouée} \end{cases}$$

Le principe est d'utiliser une table qui décrit toutes les pages d'un 2-ESPACE. Chaque entrée de cette table contient:

- soit le numéro de la page allouée dans le 1-ESPACE (N)
- soit un indicateur d'invalidité si la page n'est pas allouée.

L'éxécution de la fonction f consiste à découper l'adresse A en deux groupes de bits:

- les 19 bits de poids fort déterminent une entrée dans la table.
- les 12 bits de poids faible constituent un déplacement D dans une page.

Si l'entrée de la table obtenue est invalide la fonction f est indéterminée, sinon on en déduit le couple (N,D).

La gestion d'une telle table est peu pratique du fait de sa taille (2**19 entrées), et d'autre part du fait de l'allocation dynamique des pages. En effet un grand nombre d'entrées peuvent être

invalides ce qui occasionne une perte de place. On crée donc plusieurs niveaux de tables en découpant l'adresse en plusieurs groupes de bits. On constate que plus le nombre de niveaux de tables est grand, plus elles sont petites et leur gestion est simple. Mais en contre-partie le temps d'éxécution de la fonction f augmente avec le nombre de niveaux. Nous avons décidé de se limiter à deux niveaux. Le découpage de l'adresse A se fait en trois groupes

- les S1 bits de poids forts constituent un numéro de segment.
- les S2 bits suivants constituent un numéro de page (S1+S2=19)
- les 12 bits de poids faible constituent le déplacement D.

Chaque entrée des tables de niveau 1 (ou table de segments) contient:

- soit l'adresse d'une table de niveau 2
- soit un indicateur d'invalidité s'il n'y a pas de table associée.

Chaque entrée des tables de niveau 2 contient:

- soit le numéro de la page dans le 2-ESPACE
- soit un indicateur d'invalidité si la page n'est pas allouée

Du fait que les fichiers distribués aléatoirement dans l'espace d'adressage sont rares, nous avons décidé de minimiser les table de niveau 2 en prenant:

- S2=8 ce qui donne des tables de 1K
- S1=11 ce qui donne des tables de 8K

Remarquons que ces tailles peuvent être paramétrées ce qui permet de les changer facilement.

2) LE 2-ESPACE VU PAR L'UTILISATEUR.

2.1) 2-ESPACE mémoire virtuelle.

L'utilisateur voit le 2-ESPACE c'est à dire le fichier comme un espace adressable. Il ne se préoccupe pas du découpage en bloc physique (pages).

2.2) Primitive de lecture.

L'utilisateur peut lire une portion d'un 2-ESPACE à l'aide de:

LIRE (NOM,AM,AD,n)

avec - NOM le nom du 2-ESPACE
- AM l'adresse mémoire où on va lire
- AD l'adresse dans le 2-ESPACE
- n le nombre de caratère à lire

Cette primitive a pour effet d'amener l'information demandée. L'allocation dynamique des blocs physiques rend possible la lecture de ce qui a été écrit seulement. Lorsque l'adresse AD désigne une portion du 2-ESPACE qui n'existe pas on retourne une zone nulle.

2.3) Primitive d'écriture.

L'utilisateur peut écrire dans le 2-ESPACE à l'aide de:

ECRIRE (NOM,AD,AM,n)

dont les paramètres sont identiques à ceux de LIRE.

Lors de l'écriture il y a allocation automatique d'un ou plusieurs blocs physiques.

3) PRINCIPE DE REALISATION.

3.1) Contexte de réalisation.

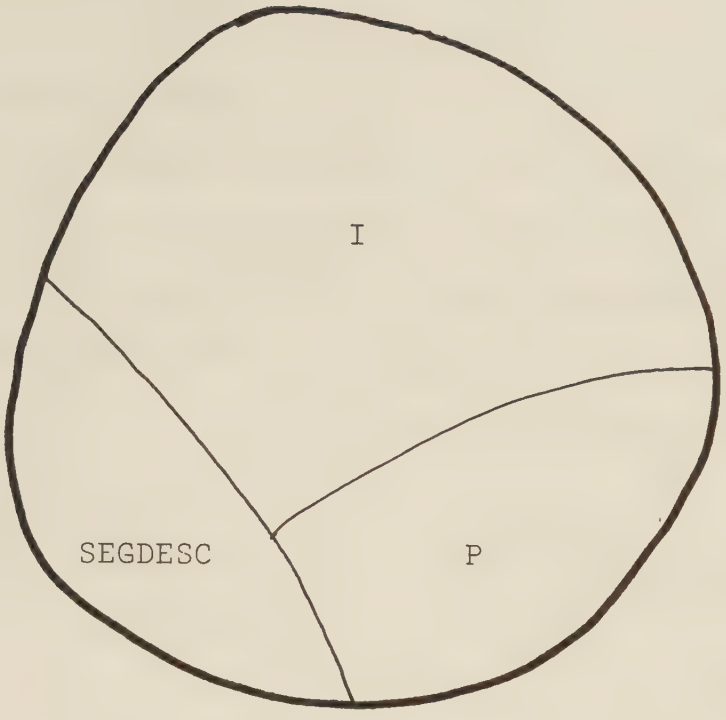
Le projet CMS+ prévoie de réécrire entièrement un nouveau système. Pour commencer l'implémentation de la gestion de fichiers il fallait donc attendre que les mécanismes de base du noyau de CMS+ soient implémentés: gestion des interruptions, gestion de la mémoire, etc... Afin de ne pas perdre de temps on a décidé de commencer la réalisation en utilisant comme système de base CMS, et d'employer les mécanismes de 1-ESPACE (chapitre 2) et de M-ESPACE (chapitre 3) déjà opérationnels. Les routines de la gestion de fichiers de CMS+ sont placées initialement en zone utilisateur ainsi que des programmes de test, et peu à peu elles seront incluses dans le noyau de CMS qui aura une gestion de fichiers entièrement nouvelle.

Les routines de la gestion de fichiers sont écrites en langage de haut niveau GSL (G1) et on utilise pour la mise au point la machine SPY (L2) ainsi que le 'debugging' symbolique adapté à GSL (S1).

3.2) Structure du contenu d'un 1-ESPACE.

Les 2-ESPACE constituent les fichiers d'un utilisateur et sont donc contenus dans un 1-ESPACE qui sert de support physique à l'information.

Sur la figure 4.1 on voit que I représente les informations stockées c'est à dire un ensemble de 2-ESPACE. Ces informations sont contenues dans des pages du 1-ESPACE et la partie P représente



1-ESPACE

SEGDESC : Segment descripteur

I : Ensemble des informations stockées
(suite de 2-ESPACE)

P : Pages libres du 1-ESPACE

Figure 4.1 Structure d'un 1-ESPACE

l'ensemble des pages non utilisées, c'est à dire des pages libres.

La description de P et de I dans le 1-ESPACE est faite à l'aide du segment descripteur (SEGDESC) qui contient les informations pour accéder aux pages des différents 2-ESPACE ainsi que le catalogue des noms de ceux-ci.

Dans la suite on admettra que le catalogue existe et qu'il est géré par un algorithme spécialisé à qui l'on fournit un nom et qui donne en retour une condition d'invalidité s'il n'existe pas ou l'adresse des informations décrivant le 2-ESPACE. Cette partie est indépendante et peut être réalisée de différentes manières. On verra au 4) une implémentation possible.

3.3) Le segment descripteur.

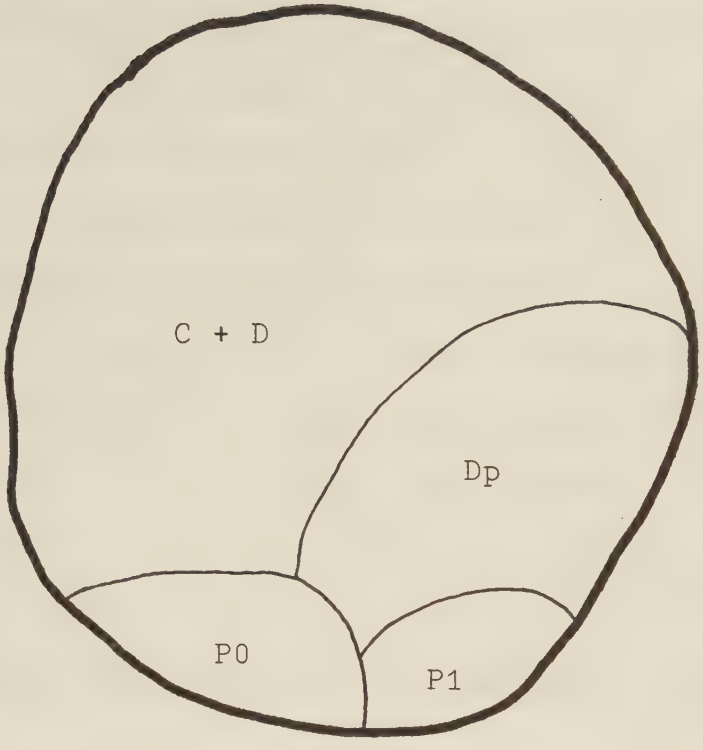
3.3.1) Définition.

Le segment descripteur (noté SEGDESC cf fig. 4.2) est la partie du 1-ESPACE qui décrit les 2-ESPACE ainsi que la provision de pages libres du 1-ESPACE lui-même.

Il contient:

- sa propre description et gestion,
- le catalogue des noms des 2-ESPACE,
- la description des 2-ESPACE,
- la gestion des pages libres du 1-ESPACE,
- les zones pour les transferts entre la mémoire virtuelle et le 1-ESPACE,
- la liste des 2-ESPACE actifs.

On utilise le terme 'segment' car on a pris la convention de le placer entièrement dans un ou plusieurs segments virtuels lorsque le 1-ESPACE est actif. Il est constitué de pages du 1-ESPACE qu'il



SEGDESC

C : Liste des noms des 2-ESPACE (le catalogue)

D : Description des 2-ESPACE (tables de pages externes)

Dp : Description de P (fig. 4.1)

P0 : Page 0 du 1-ESPACE et du SEGDESC

P1 : Page 1 du 1-ESPACE et du SEGDESC

Figure 4.2 Structure du segment descripteur

suffit d'amener en mémoire virtuelle lorsqu'il est actif.

Les références internes au segment descripteur sont faites sous forme d'adresses mémoire relatives au début du segment virtuel dans lequel il est chargé, et les références externes sont faites sous forme de numéro de page dans le 1-ESPACE. Cette technique permet de charger le SEGDESC dans un segment arbitraire en mémoire virtuelle. Il suffit d'ajouter à toutes les adresses le numéro du segment où il réside. D'autre part il est inutile de le réarranger pour le réécrire sur le 1-ESPACE car il est structuré en pages. Il est alors possible lors d'une mise à jour de recopier le SEGDESC tel qu'il est c'est à dire sans fermer les fichiers actifs, et sans libérer les zones d'entrées-sorties. Il en résulte une mise à jour plus rapide.

3.3.2) Description des pages composant le segment descripteur.

La suite des pages composant le segment descripteur (SEGDESC) a deux lieux de résidence: en mémoire et sur le 1-ESPACE. Il faut donc disposer d'une table de pages externes qui permette de faire la correspondance entre la mémoire et le 1-ESPACE.

On remarque que cette table doit être à un emplacement fixe dans le 1-ESPACE car l'accès au SEGDESC n'est possible que par son intermédiaire. La solution adoptée est d'employer la première page du 1-ESPACE (page 0 notée P0) qui a pour correspondante en mémoire la première page du segment dans lequel on décide de charger le SEGDESC.

Le nième mot (4 octets) de P0 décrit la nième page du SEGDESC en mémoire. Il peut contenir (cf fig. 4.3):

- une valeur V>0 du mot indique que la nième page du SEGDESC existe et a pour correspondante dans le 1-ESPACE la page numéro V.
- une valeur V<0 du mot indique que la nième page du SEGDESC est

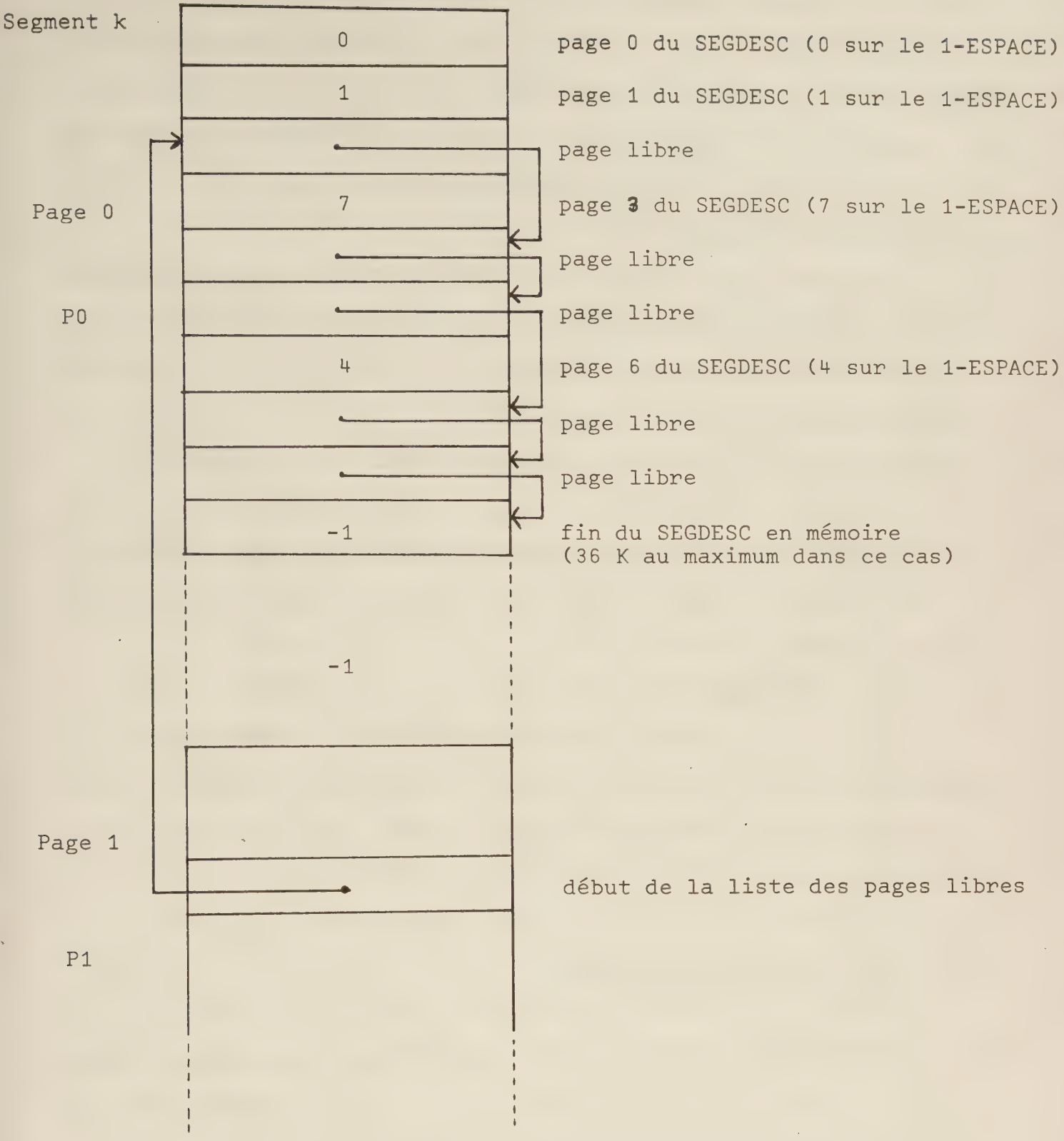


Figure 4.3 Description des pages du SEGDESC

libre et la valeur absolue de V indique l'adresse du prochain mot qui désigne à son tour une page libre. Les mots représentant des pages libres sont donc chainés entre eux. Le début de cette liste est placé dans la page numéro 1 (notée P1) qui joue aussi un rôle particulier (elle contient des constantes dont nous verrons l'utilité par la suite).

- une valeur V=-1 du mot indique que la nième page du SEGDESC en mémoire n'existe pas. En effet on utilise la possibilité qu'ont les M-ESPACE (chapitre 3) de distribuer de la mémoire de manière discontinue au niveau du segment. Sur la figure 4.3 le segment en mémoire comporte 9 pages soit 36K au maximum.

La page P0 contient 1024 mots ce qui autorise une taille maximum de 1024 pages soit 4 segments virtuels consécutifs pour le SEGDESC.

La lecture du SEGDESC depuis le 1-ESPACE en mémoire se fait donc en deux étapes:

- on lit la page P0

- on parcourt les mots de P0 et chaque fois qu'on rencontre une valeur V>0 on lit en mémoire la page numéro V du 1-ESPACE à l'aide de la primitive LIREPAGE (cf chapitre 2 2.2), qui rappelons-le ne provoque aucune entrée-sortie réelle.

3.3.3) Description d'un 2-ESPACE.

Comme on l'a vu, le 2-ESPACE est un espace adressable jusqu'à l'octet. Toute adresse est découpée en trois groupes (cf 1.3) comportant respectivement 11, 8, et 12 bits:

S1,S2,D

avec $0 \leq D \leq 4095$

$0 \leq S1 \leq 2047$

$0 \leq S2 \leq 255$

On utilise deux sortes de tables (niveau 1 et niveau 2) qui permettent de déterminer dans quelle page du 1-ESPACE se trouve l'information référencée. L'ensemble constitue la table de pages externes d'un 2-ESPACE.

La table externe a deux formats possible:

- table externe compactée (notée TEC) qui tient le minimum de place.

- table externe étendue (notée TEE) utilisée lorsque le 2-ESPACE est actif pour traduire les adresses le plus rapidement possible.

a) Table externe compactée (TEC).

La TEC décrit la suite des pages composant le 2-ESPACE. Elle est constituée de doublets:

(numéro de page dans le 1-ESPACE, nombre de pages)

Chaque doublet décrit une suite de pages contigues.

Ex: soit le 2-ESPACE comprenant les pages:

10,11,20,25,26,27

La TEC sera: (10,2),(20,1),(25,3)

Du fait que le 2-ESPACE est un espace d'adressage et qu'il y ait une allocation dynamique des pages, l'utilisateur peut provoquer des discontinuités.

Ex: écrire à l'adresse 0 provoque l'allocation d'une page

écrire à l'adresse 40960 provoque l'allocation d'une deuxième page.

Il faut se rappeler que la deuxième page allouée n'est pas contiguë à la précédente dans le 2-ESPACE. En effet l'adresse 40960 est située dans la dixième page du 2-ESPACE. Entre les deux il y a donc 8 pages non allouées qui doivent figurer dans la description. On utilise un doublet spécial:

(0,P)

L'exemple ci-dessus donnera comme description:

(68,1),(0,8),(74,1) pages 68, 74 dans le 1-ESPACE

En pratique les doublets constituant la TEC sont regroupés dans des blocs (Description 2-ESPACE Bloc noté D2B fig. 4.4) comprenant deux pointeurs qui les chainent entre eux, et 7 doublets (chaque bloc a donc une taille de 64 octets, ce qui est compatible avec une gestion de mémoire de type BUDDY SYSTEM cf 3.3.5).

Les derniers doublets non utilisés (dans le dernier D2B) contiennent (0,0) ce qui indique qu'ils sont libres.

L'ensemble des D2B constitue la TEC.

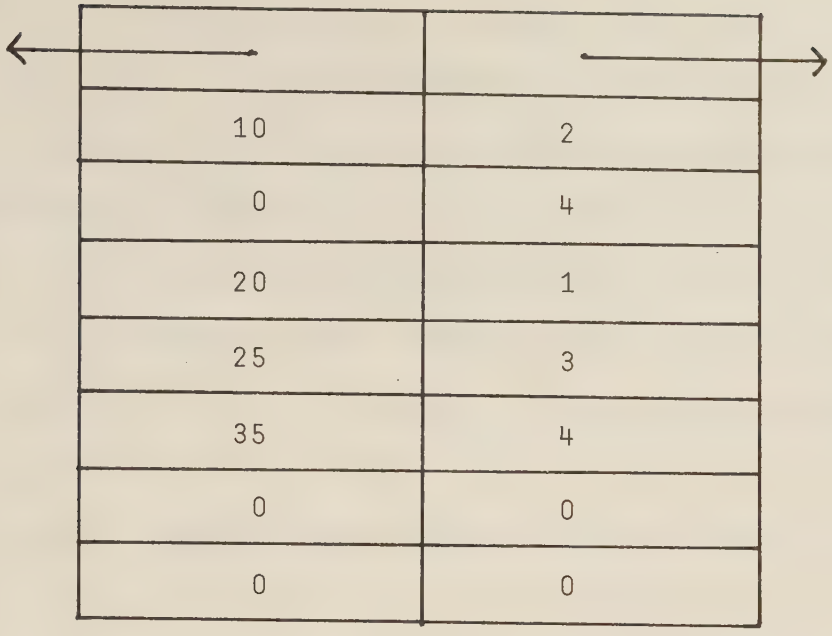
b) Table externe étendue (TEE).

Lorsque le 2-ESPACE est actif il faut déterminer le plus rapidement possible dans quelle page du 1-ESPACE se trouve l'information référencée par l'utilisateur. La structure de la TEC impose une recherche en liste qui est d'autant plus longue que l'élément recherché a une adresse élevée.

A l'activation du 2-ESPACE la TEC est étendue en une table externe étendue qui comprend autant d'entrées que le 2-ESPACE comporte de pages. La TEE est constituée de tables de rang 1 et de rang 2 (cf 1.3). Les premières jouent un rôle équivalent à des tables de segments, les secondes à des tables de pages. On a donc une description analogue aux mémoires virtuelles classiques.

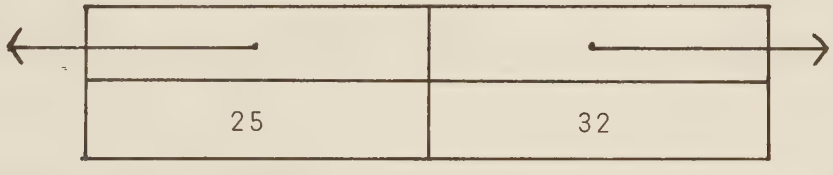
La table de segment comprend par entrée:

- l'adresse de la table de pages associées
- le nombre de pages associées
- un indicateur d'invalidité s'il n'y a pas de tables de pages associées



D2B

Figure 4.4 Structure du D2B



mini-D2B

Figure 4.5 Structure du mini-D2B

La table de segment peut avoir au maximum 2048 entrées (cf 1.3) soit une longueur de 8K. En fait à l'initialisation on lui alloue deux mots, et on double sa taille (avec recopie globale) chaque fois que l'emplacement précédent est insuffisant.

Une table de page comprend par entrée:

- soit une valeur V>0 désignant la page numéro V du 1-ESPACE
- soit une valeur V<0 désignant une page non allouée

Chacune peut avoir 256 entrées au maximum, mais on les initialise à deux mots et cette taille est doublée quand elle est insuffisante.

L'ensemble de ces tables constituent la TEE. C'est elle qui permet de calculer la fonction:

adresse∈2-ESPACE⟼(numéro de page∈1-ESPACE,déplacement)

On remarque donc qu'il faut au maximum trois interruptions pour page manquante pour accéder au 2-ESPACE quelque soit la structure de celui-ci, et l'état d'allocation des pages du 1-ESPACE.

Le 2-ESPACE de taille minimum est décrit par:

- une table de segment de deux mots
- une table de page de deux mots

soit quatre mots au total.

Le 2-ESPACE de taille maximum (524288 pages) est décrit par:

- une table de segment de 2048 entrées soit 8K
- 2048 tables de pages de 256 entrées soit 2048K

La description du fichier maximum peut donc être contenue dans le SEGDESC (4 segments en mémoire au maximum) ce qui est cohérent.

3.3.4) Gestion des pages libres du 1-ESPACE.

L'ensemble des pages non utilisées du 1-ESPACE peuvent être décrites de la même manière qu'un 2-ESPACE à l'aide de doublets:

(numéro de page dans le 1-ESPACE, nombre de pages)

En pratique l'utilisation de D2B s'avère délicate car l'insertion d'un doublet supplémentaire est compliquée. On utilise donc une liste de mini-D2B comportant un seul doublet ce qui facilite leur gestion (cf fig. 4.5). Le point d'ancrage de cette liste est dans la page 1 (P1) de même que le mini-D2B initial lorsque tout le 1-ESPACE est vide. Remarquons que les mini-D2B sont gérés de façon spéciale pour qu'ils soient tous regroupés dans des pages connues (utile pour la mise à jour du SEGDESC).

On dispose de deux fonctions de gestion des pages du 1-ESPACE:

a) DEMPAGE (N,S)

Cette fonction permet d'acquérir une page sur le 1-ESPACE. Le numéro de la page attribuée est placé en retour dans N. S désigne le numéro du segment en mémoire où réside le SEGDESC du 1-ESPACE considéré (au cas ou plusieurs 1-ESPACE sont actifs).

L'éxécution de cette fonction consiste à parcourir la liste des mini-D2B en la réorganisant si nécessaire.

b) LIBPAGE (N,S)

Cette fonction libère la page numéro N du 1-ESPACE dont le SEGDESC est dans le segment virtuel S.

Si la page est contiguë à d'autre pages libres on modifie uniquement le mini-D2B considéré. Sinon on crée un mini-D2B qui est inséré dans la chaine.

Ces deux fonctions tiennent à jour un compteur (NPL) dans P1 qui contient le nombre de pages libres du 1-ESPACE.

3.3.5) Gestion de la mémoire libre du segment descripteur

La gestion de la mémoire libre du SEGDESC est faite au niveau des pages et au niveau du contenu des pages. Toutes les adresses sont des déplacements relatifs au début du segment virtuel en mémoire.

a) Gestion du contenu des pages.

On utilise une gestion de type BUDDY SYSTEM (K1). Le principe est d'allouer des blocs dont la taille est un puissance de 2, en découpant en 2, en 4 etc... le plus grand bloc disponible (ici la page de 4096 octets soit 2**12).

Le fait d'utiliser des pages donne une propriété fondamentale aux blocs: leur adresse est toujours un multiple de leur taille. Ceci va permettre de simplifier au maximum la restructuration des blocs lors de la libération de mémoire.

La structure d'un bloc libre est la suivante:

- un indicateur permettant de savoir que le bloc est libre
- la taille du bloc
- un pointeur de chainage des blocs libres de même taille

L'ancrage des différentes listes de blocs libres (cf fig. 4.6) est faite dans une table TBUDDY placée dans P1, qui contient autant d'entrée que de tailles possibles. On remarque que les demandes égales ou supérieures à une page ne peuvent être satisfaites à ce niveau (cf b)).

La demande de mémoire se fait à l'aide de la fonction:

GETBLOC (T,AM,S)

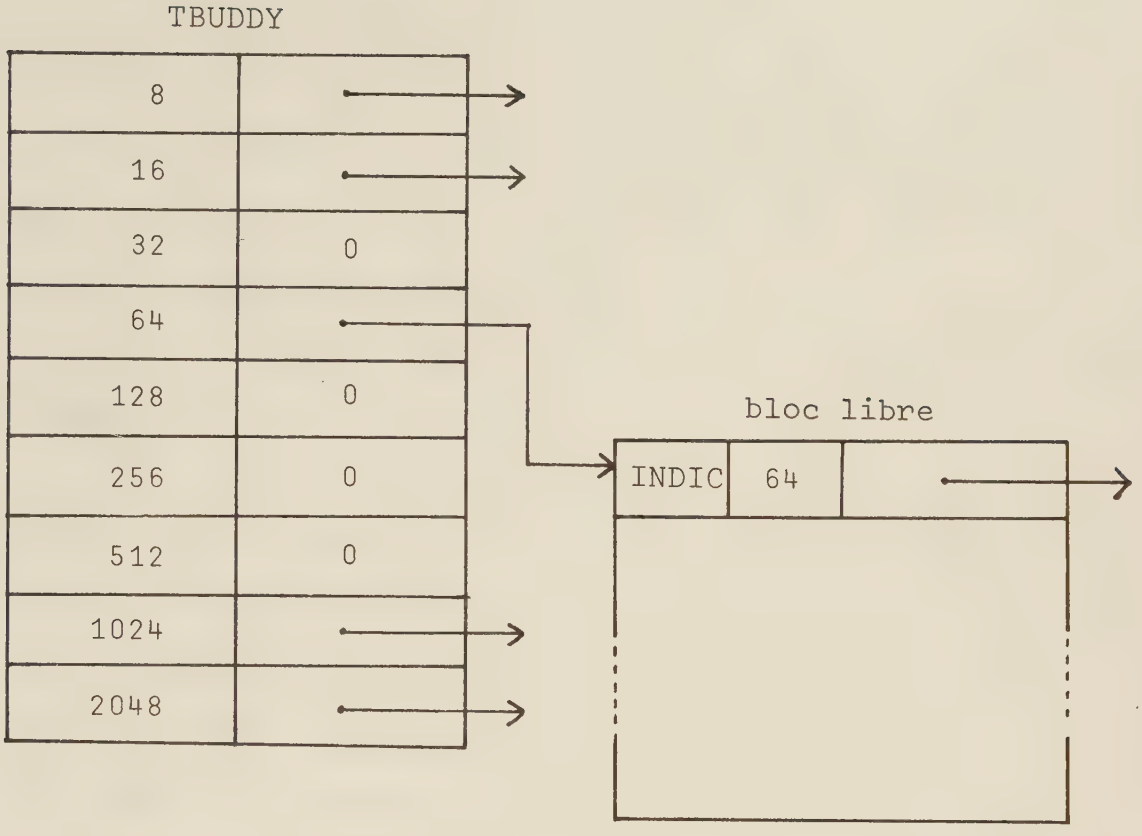


Figure 4.6 Gestion de la mémoire libre du SEGDESC

avec en entrée:

T taille du bloc demandé

S numéro du segment où réside le SEGDESC

et en sortie:

AM adresse du bloc

A l'éxécution on recherche dans la table TBUDDY s'il existe un bloc de taille T (en fait la puissance de 2 immédiatement supérieure ou égale). Si oui on l'enlève de la liste, ainsi que l'indicateur 'libre' et on le donne à l'utilisateur.

S'il n'existe pas de bloc de taille T on réappelle la fonction récursivement avec une taille double:

GETBLOC (2T,AM,S)

Au retour on partage le bloc obtenu en deux: une moitié est chainée comme bloc libre dans la table TBUDDY, l'autre moitié est donnée à l'utilisateur (ou au niveau de récursion inférieur).

Lorsqu'on arrive à une demande dont la taille est celle d'une page (T=4096) on s'adresse aux fonctions qui gèrent les pages du SEGDESC (GETPAGE cf b)).

La libération de mémoire se fait à l'aide de la fonction:

FREEBLOC (T,AM,S)

avec T taille du bloc à libérer

AM adresse du bloc à libérer

S numéro du segment où réside le segment descripteur

A l'éxécution on teste d'abord la validité des paramètres. En effet ils doivent satisfaire:

AM//T=0 (// reste de la division)

Ensuite on regarde si la restructuration du bloc qu'on libère avec ses voisins est possible. Pour cela on essaie de libérer le bloc actuel ainsi qu'un bloc adjacent de même taille, c'est à dire d'en libérer un de taille double. Remarquons que le bloc de taille

double doit conserver la propriété fondamentale du BUDDY SYSTEM: l'adresse doit être un multiple de la taille. Il en résulte que le bloc adjacent que l'on peut libérer est unique: on l'appelle bloc adjacent associé (BUDDY).

L'adresse du bloc adjacent associé se détermine de la manière suivante:

- si AM//2T=0 alors le bloc adjacent associé suit le bloc qu'on libère et a pour adresse AM+T

- si AM//2T≠0 alors le bloc adjacent associé précède le bloc qu'on libère et a pour adresse AM-T

On teste ensuite si le bloc adjacent associé est libre et s'il est de même taille. On utilise pour cela l'indicateur et la taille mémorisés dans les blocs libres ce qui évite de parcourir une liste d'ou un gain de temps. Si ces deux conditions ne sont pas remplies la restructuration est impossible et on chaine le bloc à libérer dans la table TBUDDY.

Si les deux conditions sont remplies on peut libérer un bloc de taille double, et on réappelle la fonction récursivement:

FREEBLOC (2T,AM,S) si le bloc adjacent associé suit le bloc

FREEBLOC (2T,AM-T,S) si le bloc adjacent associé précède le bloc

Lorsque la taille devient égale à celle d'une page (T=4096) on s'adresse aux fonctions qui gèrent les pages du SEGDESC (FREEPAGE).

On constate donc qu'une telle gestion de mémoire est rapide. Elle présente cependant un inconvénient: toute demande de mémoire provoque l'allocation d'un bloc dont la taille est une puissance de 2 obtenue en arrondissant la taille demandée. Si les tailles demandées sont aléatoires il peut en résulter une perte de place. Mais si toutes les tailles sont des puissances de 2 (comme dans CMS+) le coéfficient de remplissage est très bon.

b) Gestion des pages du segment descripteur.

Comme on l'a vu la page 0 (P0 première du segment en mémoire et sur le 1-ESPACE) décrit les pages du SEGDESC. Lorsque le nième mot contient une valeur V<0 la nième page est libre (cf 3.3.2) et la valeur absolue de V indique l'adresse du prochain mot désignant une page libre. La gestion des pages libres du SEGDESC se fait donc en parcourant cette liste.

Remarquons que la gestion des pages libres du SEGDESC est liée à celle des pages du 1-ESPACE (cf 3.3.5). En effet toute allocation de page en mémoire pour le SEGDESC doit s'accompagner d'une allocation sur le 1-ESPACE pour être sûr d'avoir la place de le recopier. On tient donc à jour un compteur (NEWPL en page 1 P1) qui est augmenté de 1 à chaque allocation de page et qui est annulé lorsque le SEGDESC est mis à jour sur le 1-ESPACE.

La demande de pages se fait à l'aide de la fonction:

GETPAGE (N,AM,S)

avec N nombre de pages contigues demandées en mémoire virtuelle (en général N=1, ou N=2 pour les TEE)

AM adresse mémoire de la première page en retour

S numéro du segment où réside le SEGDESC

A l'éxécution on recherche en page 0 N mots consécutifs désignant des pages libres. Si cette opération s'avère impossible la mémoire disponible pour le SEGDESC est saturée et l'éxécution est interrompue (il est possible d'utiliser les propriétés d'extension dynamique des segments des M-ESPACE décrites au chapitre 3).

Lorsqu'on trouve N mots consécutifs, on réorganise la chaine et on demande parallèlement N pages non forcément contigues dans le 1-ESPACE en émettant N fois la fonction DEMPAGE (cf 3.3.4). Les

numéros des pages du 1-ESPACE sont alors placés dans les N mots consécutifs de la page 0.

En fait aprés chaque éxécution de la fonction DEMPAGE on augmente de 1 le compteur NEWPL (nombre de nouvelles pages attribuées au SEGDESC), et on le compare au compteur NPL (nombre de pages libres). Si:

NPL>NEWPL

alors on continue d'allouer des pages. Par contre si:

NPL=NEWPL

ceci signifie que le 1-ESPACE est saturé et que l'on a juste la place de réécrire le SEGDESC. On donne le contrôle à une routine de sauvegarde instantanée qui ferme les fichiers actifs, réécrit le SEGDESC sur le 1-ESPACE et interrompt l'éxécution.

La libération des pages se fait à l'aide de la fonction:

FREEPAGE (N,AM,S)

avec N nombre de pages contigues à libérer

AM adresse de la première page

S numéro du segment où réside le SEGDESC

A l'éxécution l'adresse permet de déterminer en page 0 la suite de N mots à inclure dans la liste des pages libres. Leur contenu donne le numéro des pages du 1-ESPACE qu'il faut libérer à l'aide de la fonction LIBPAGE. La suite des mots de la page 0 est réorganisée.

On avertit CP+ pour qu'il libère les pages dont on n'a plus besoin dans l'espace de pagination.

On diminue de 1 le compteur NEWPL.

3.3.6) Zones d'entrées-sorties.

Comme on l'a vu le 2-ESPACE est une mémoire adressable. L'accès

au nième octet se fait en déterminant la page du 1-ESPACE dans lequel il réside et en lisant cette page en entier. Il est donc nécessaire d'utiliser des zones d'entrées-sorties qui soient placées dans le SEGDESC.

Le fait d'utiliser la pagination impose quelques restrictions: une page d'un 1-ESPACE est obligatoirement lue en mémoire dans une page dont l'adresse est alignée sur un multiple de 4096 (et non pas dans un bloc de 4096 octets placé à une adresse arbitraire).

Lorsqu'un utilisateur demande la lecture d'une portion d'un 2-ESPACE à l'aide de la fonction LIRE (cf 2.2) et que le nombre de caractères demandés est plus petit que 4096, la ou les pages contenant l'information sont lues dans une zone d'entrées-sorties et l'information est transférée dans la mémoire de l'utilisateur. Par contre si le nombre de caractères demandés est plus grand que 4096, la lecture directe dans la mémoire de l'utilisateur n'est possible que s'il y a compatibilité entre les frontières de pages du 1-ESPACE et l'adresse mémoire fournie par l'utilisateur.

Ceci pose un problème par exemple pour les fichiers de type 'MODULE' qui ne peuvent pas être chargés directement dans la mémoire de l'utilisateur s'il y a incompatibilité d'adresse. Cependant ce cas particulier peut être résolu par les commandes qui gèrent les modules (GENMOD,LOADMOD). Dans les autres cas (relativement rare pour des enregistrements supérieurs à 4096 octets) l'utilisateur devra gérer efficacement sa mémoire pour optimiser les transferts avec le 2-ESPACE.

3.3.7) Mise à jour du segment descripteur.

La mise à jour du SEGDESC consiste à réécrire son contenu sur le 1-ESPACE. Cette opération est faite par CMS+ en fin de commande par

exemple, ou par l'utilisateur ('CHECK-POINT RESTART').

Pour des raisons de sécurité on ne réécrit pas le SEGDESC directement dans les pages du 1-ESPACE qui lui sont allouées. En effet si pour une raison (hardware ou software) la mise à jour était interrompue, le SEGDESC composé d'une partie mise à jour et d'une partie non mise à jour pourrait être incohérent. L'accès au 1-ESPACE serait alors impossible.

On détecte les pages du SEGDESC qui ont été modifiées à l'aide de la primitive:

TESTPAGE (AD,AM,N) (cf chapitre 2 2.4)

Seul ces pages sont réécrites sur le 1-ESPACE à un nouvel emplacement. A cette étape l'ancienne copie du SEGDESC est toujours disponible. Ensuite on acquiert un nouvel emplacement pour les pages qui contiennent les mini-D2B (cf 3.3.4). On libère les anciennes copies des pages du 1-ESPACE attribuées au SEGDESC, et on recopie les pages contenant les mini-D2B. Enfin on modifie les pages 0 et 1, et leur recopie valide le nouveau SEGDESC. Le moment où la réécrirure est critique se réduit à deux pages.

On remarque qu'il n'est pas nécessaire de modifier le SEGDESC avant de le réécrire. La mise à jour peut donc être faite sans fermer les fichiers, les zones d'entrées-sorties étant directement écrites dans les fichiers concernés. Il est bien évident que dans le cas d'une mise à jour faite normalement par le système on fermera les fichiers ce qui réduit la taille du SEGDESC.

La procédure de sauvegarde instantanée évoquée au 3.3.5 ne procède pas de la même manière. En effet le nombre de pages libres du 1-ESPACE est égal au nombre de nouvelles pages attribuées au SEGDESC. On ne peut donc pas réécrire les pages modifiées à un nouvel emplacement. Il faut prendre le risque de réécrire le SEGDESC directement dans les pages du 1-ESPACE qui lui sont allouées.

3.4) Opérations sur un 2-ESPACE.

3.4.1) Ouverture-fermeture.

Ces deux opérations sont rendues nécessaires pour mettre les tables décrivant le 2-ESPACE sous une forme qui permette un accès rapide. Remarquons qu'elles peuvent être éxécutées automatiquement par CMS+: l'ouverture lorsque l'utilisateur emet les primitives LIRE ou ECRIRE (cf 2.2 2.3), la fermeture lorsque CMS+ décide de mettre à jour le segment descripteur.

a) Ouverture

On recherche le nom du 2-ESPACE dans le catalogue (cf 4.) et en retour on a soit l'indication que le nom n'existe pas, soit l'adresse d'un bloc décrivant le 2-ESPACE qui contient l'adresse de la table externe compactée (TEC) ainsi que les caractéristiques du 2-ESPACE (format des enregistrements fixe ou variable, nombre d'enregistrements, nombre de pages, etc...).

L'opération d'ouverture consiste à étendre la table externe compactée en une table externe étendue (TEE), c'est à dire sous forme d'une table de segment et d'une ou plusieurs tables de pages.

On construit une table d'extension pour 2-ESPACE ouvert (EXFO) chainée au bloc le décrivant. Elle contient:

- l'adresse de la TEE
- un indicateur validant la TEC
- le nombre de segment du 2-ESPACE
- l'adresse de la zone d'entrées-sorties (une page)
- le numéro de la page du 1-ESPACE dont la zone d'entrée-sortie

représente une copie.

On conserve la TEC car si on ne modifie pas la configuration des pages du 2-ESPACE (cas d'une lecture simple) la TEC reste inchangée (indicateur dans l'EXFO).

b) Fermeture.

La fermeture du 2-ESPACE consiste à recopier la zone d'entrées-sorties dans le 2-ESPACE, et à libérer l'EXFO. Si la TEC est toujours valide on se contente de libérer la TEE. Sinon on reconstruit une nouvelle TEC à partir de la TEE et on libère cette dernière.

3.4.2) Primitives d'accès.

a) Primitive LIRE.

Son format est (cf 2.2):

LIRE (NOM,AM,AD,n)

avec NOM nom du 2-ESPACE

AM adresse mémoire où on va lire

AD adresse dans le 2-ESPACE

n nombre de caractères à lire

Comme on l'a vu (cf 1.3) l'adresse AD est découpée en trois groupes de bits:

S1,S2,D

On indexe successivement la table de segment, et une table de page. On obtient soit une impossibilité et on retourne n octets nuls à l'utilisateur, soit un doublet (N,D).

On regarde s'il existe une zone d'entrées-sorties en mémoire (adresse dans l'EXFO), et si elle représente déjà la page du 1-ESPACE concernée: N. Si oui la page est en mémoire et on peut

transférer l'information directement à l'utilisateur. Sinon on acquiert une zone d'entrées-sorties si nécessaire, et on lit la page N à l'aide de la primitive LIREPAGE (cf ch 2 2.2). On place N dans l'EXFO, et on donne l'information à l'utilisateur.

Si n>4096 on regarde s'il est possible de lire directement dans la mémoire de l'utilisateur (compatibilité entre les adresses mémoire et frontière de pages). Si c'est impossible la lecture passe par l'intermédiaire d'une zone d'entrées-sorties.

b) Primitive ECRIRE.

Son format est (cf 2.3):

ECRIRE (NOM,AM,AD,n)

dont les paramètres sont identiques à LIRE.

De la même manière à partir de AD on obtient soit une condition d'invalidité, soit un doublet (N,D). Dans le premier cas on construit une nouvelle table de pages si nécessaire, et on alloue une page supplémentaire au 2-ESPACE. Dans les deux cas on obtient donc un doublet (N,D).

Ensuite on teste s'il existe une zone d'entrées-sorties en mémoire et si elle représente la page numéro N. Si oui on se contente d'y transférer l'information. Si non on réécrit la zone d'entrées-sorties éventuelle dans le 1-ESPACE. Le même emplacement mémoire peut à nouveau servir comme zone d'entrées-sorties représentant la page N, et on y transfert l'information.

Si n est tel que l'information ne tient pas dans la zone d'entrées-sorties, on découpe l'information (enregistrement 'à cheval' sur deux pages). Si n>4096 on essaie d'écrire directement depuis la mémoire de l'utilisateur, si c'est impossible on fait autant de transfert que nécessaire.

3.5) Opérations sur un 1-ESPACE.

3.5.1) Accés à un 1-ESPACE.

L'accès à un 1-ESPACE se fait par la commande de CMS+ 'ACCESS' qui peut être émise soit automatiquement par le système pour un 1-ESPACE d'adresse implicite (comme le disque P de CMS), soit par l'utilisateur. Son format est:

ACCESS CUU (adresse du 1-ESPACE)

De façon interne, CMS+ recherche un segment libre de numéro i pour y placer le segment descripteur. Ensuite CMS+ lit la page 0 du 1-ESPACE dans la première page du segment i, teste si la taille mémoire disponible est compatible avec celle du SEGDESC et lit les pages si c'est possible.

Dans l'hypothèse où l'on admet le partage des 1-ESPACE entre plusieurs utilisateurs, le segment descripteur doit être partagé entre eux. Seul CP+ est capable de réaliser cette opération (LIER cf chapitre 3). L'opération d'ACCESS doit donc s'accompagner d'une instruction DIAGNOSE:

DIAGACCESS (CUU,i)

avec CUU adresse du 1-ESPACE

i numéro de segment où va résider le SEGDESC

Dans la FTABLE (cf ch 2) associée au 1-ESPACE d'adresse CUU on doit disposer d'un pointeur supplémentaire (PTSEGDESC) qui donne l'adresse de la table de page décrivant le segment descripteur en mémoire. A l'initialisation ce pointeur est nul. Il se pose donc le problème du premier utilisateur.

a) Premier utilisateur.

Sur la figure 4.7, lorsque USER1 emet l'instruction DIAGACCESS,

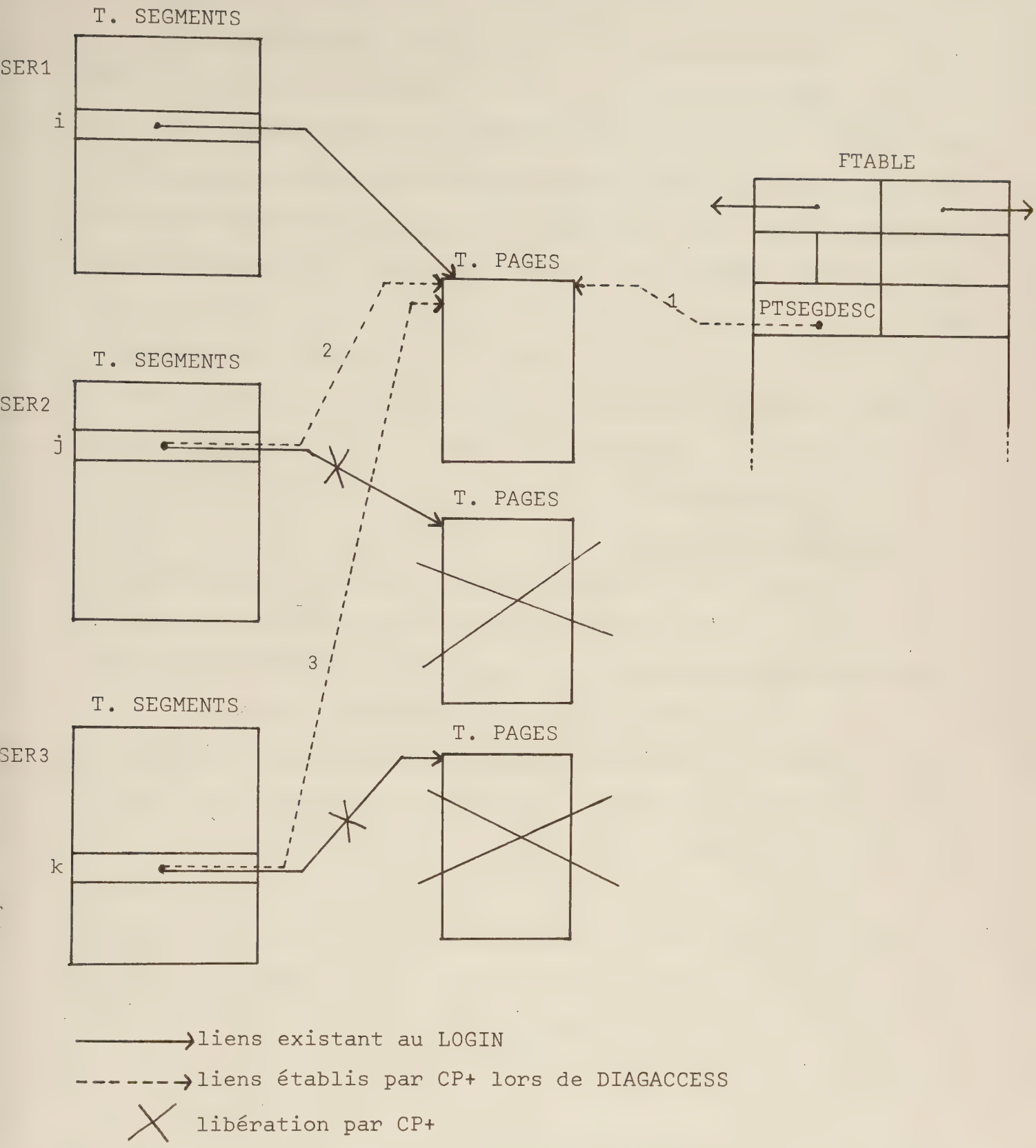


Figure 4.7 1-ESPACE partagé

CP+ trouve le pointeur PTSEGDESC nul. Il établit la liaison 1. Ensuite il est nécessaire de lire le segment descripteur.

Une première solution consiste à faire éxécuter cette lecture par l'utilisateur (à l'aide de LIREPAGE). Au retour de l'instruction DIAGACCESS, CP+ peut renvoyer un code indiquant que la lecture est nécessaire. Cette solution présente un inconvénient majeur: on ne peut pas faire confiance à l'utilisateur et on ne serait jamais sûr que la lecture a eu lieu.

Une deuxième solution consiste à faire éxécuter la lecture par CP+. Cette solution est beaucoup plus sûre, mais on remarque que CP+ doit connaitre la structure du segment descripteur: on accroit encore les liens entre CP+ et CMS+.

b) utilisateur suivant.

Sur la figure 4.7 lorsque USER2 emet l'instruction DIAGACCESS, CP+ connait par l'intermédiaire de PTSEGDESC l'adresse de la table de pages du segment descripteur en mémoire. Il éxécute la fonction:

LIER (j,USER2,i,USER1) (cf chapitre 3 6.3)

ce qui revient à libérer la table de page associée au segment j de USER2 et établir le lien 2. Le segment descripteur est déjà en mémoire, aucune lecture n'est nécessaire.

On remarque qu'au moment où les utilisateurs envoient les commandes ACCESS le protocole d'accord est déjà résolu. En effet ils doivent avoir le 1-ESPACE en commun, et c'est au niveau du catalogue de description des environnements (ou la commande LINK avec mot de passe) qu'est résolue la protection.

3.5.2) Initialisation d'un 1-ESPACE.

L'initialisation d'un 1-ESPACE se réduit à initialiser le segment descripteur et en fait ses deux premières pages P0, P1.

Dans P0 on place:

- premier mot 0 qui représente la page 0 du 1-ESPACE
- deuxième mot 1 ..... 1 du 1-ESPACE
- les mots suivant désignent des pages libres et sont chainés entre eux suivant la taille mémoire dont on dispose, et les derniers contiennent -1. Remarquons qu'à chaque opération d'ACCESS, la suite des mots contenant -1 est recalculée en fonction de la taille mémoire dont on dispose (qui peut varier).

Dans P1 on place:

- l'adresse du début de la liste des mots en page 0 désignant des pages libres
- le mini-D2B décrivant les pages libres du 1-ESPACE
  (elles sont toutes libres sauf les deux premières)
- la table TBUDDY dont toutes les entrées sont nulles
- compteur NEWPL initialisé a 0
- compteur NPL donnant le nombre de pages libres du 1-ESPACE
- l'adresse du catalogue (racine) initialisée à 0

et éventuellement d'autres paramètres utiles par la suite.

4) LE CATALOGUE.

4.1) Aspect externe.

Comme dans de nombreux systèmes (MULTICS, TSS ref. D3, D2, O1), on utilise pour CMS+ un catalogue organisé en arbre (L1). Cette technique déjà ancienne (D2), présente l'avantage de pouvoir classer ses fichiers par centre d'intérêt.

L'arbre est organisé de la manière suivante:

-il y a une racine physique (d'adresse connue)

-chaque noeud a un nom et constitue un catalogue:il contient les noms des noeuds ou feuilles du niveau inférieur

-les feuilles constituent les descripteurs de fichiers

Le nom d'un fichier est par définition un chemin de l'arbre, c'est à dire une suite de noms de noeuds séparés par un '.'.

Ex: (cf fig. 4.8)

A.B.C.D

A.B.F.H.D

Pour des raisons d'efficacité on impose que les noeuds fils d'un même père aient des noms diffèrents pour faciliter les recherches.

En cas de partage d'un 1-ESPACE il est possible de donner aux divers utilisateurs des privilèges d'accès aux fichiers. Il suffit d'adjoindre à chaque noeud la liste des utilisateurs habilités à employer les fichiers des niveaux inférieurs. Ainsi sur la figure 4.8 on voit que 'DURAND' a accès à tous les fichiers alors que 'DUPOND' ne peut accéder qu'aux fichiers:

A.B.F.G

et A.B.F.H.D

L'utilisateur dispose de fonctions qui lui permettent de

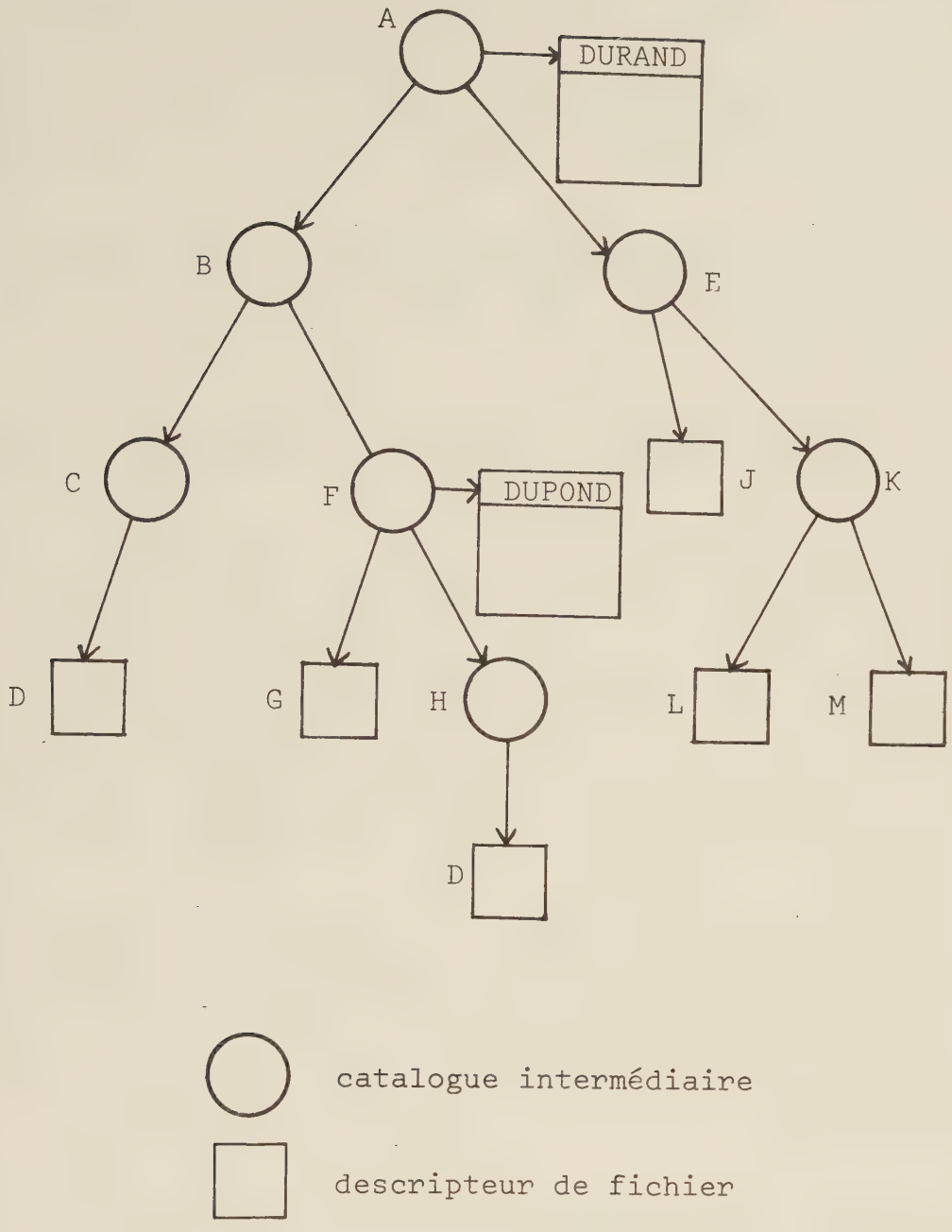


Figure 4.8 Catalogue en arbre

modifier le catalogue (création, effacement de fichiers), de le lister, ou d'autoriser le partage.

Ex: ACCES A.E MARTIN

donne l'accès à l'utilisateur à un ou plusieurs fichiers. Sur la figure 4.8, cela revient à placer 'MARTIN' dans la liste d'accès du noeud E.

D'autre part il est possible de définir une racine logique en définissant un préfixe:

Ex: PREFIXE A.B.F

permet à l'utilisateur 'DUPOND' de nommer directement ses fichiers par:

.G pour A.B.F.G

.H.D pour A.B.F.H.D

Le point permet de disposer à la fois du nom complet ou incomplet. En effet si un nom commence par un point, on le recherche par rapport à la racine logique, sinon par rapport à la racine physique.

Comme on l'a vu le catalogue est réalisable de différentes manières et même si on désire que son aspect logique soit un arbre l'aspect interne peut être différent. On va donc voir deux méthodes possibles de réalisation.

4.2) Réalisation sous forme d'arbre.

4.2.1) Représentation de l'arbre.

Dans cette approche on représente l'arbre en mémoire tel qu'il est au niveau logique. On utilise quatre types de bloc qui sont tous de taille fixe (64 octets) pour faciliter la gestion mémoire.

a) Bloc décrivant un noeud (type 1). Il contient:

- le pointeur vers le noeud père (0 si c'est la racine)
- le type du bloc (1)
- le pointeur vers la liste d'accès
- le nom du noeud
- le pointeur vers l'extension (s'il y a plus de trois branches)
- trois fois nom et adresse des noeuds fils

b) Bloc d'extension d'un noeud (type 2). Il contient:
- le pointeur vers l'extension précédente (ou le noeud)
- le pointeur vers l'extension suivante s'il y a lieu
- le type du bloc (2)
- 4 fois nom et adresse des nœuds fils

c) Descripteur de fichier (feuilles, type 3). Il contient:
- le pointeur vers le noeud père
- le type du bloc (3)
- le nom de la feuille
- les caratéristiques du fichier
- le pointeur vers la table externe compactée (TEC)

d) Bloc des listes d'accès (type 4). Il contient:
- le pointeur vers la liste précédente (ou le noeud)
- le pointeur vers la liste suivante (extension)
- le type du bloc (4)
- 4 fois nom et privilèges d'accès d'un utilisateur

4.2.2) Inconvénients de cette méthode.

Le principal inconvénient de cette méthode est l'aspect rigide de l'arbre. Les fichiers sont organisés suivant une relation unique et l'utilisateur doit s'y conformer. On peut donc envisager d'assouplir la manière de relier les fichiers entre eux.

On remarque d'autre part que s'il est facile de donner l'accès à un groupe de fichiers attachés à un même noeud, il est beaucoup plus compliqué de donner l'accès à tous les fichiers ayant un même nom. Par exemple donner l'accès à tous les fichiers dont le deuxième nom est 'TEXT', opération trés simple sous CMS (LOGIN * TEXT), reviendrait à placer le nom de l'utilisateur concerné dans toutes les listes d'accès associées aux feuilles de nom 'TEXT'. Ceci occasionnerait la création d'un grand nombre de blocs d'où une perte de place mémoire.

Enfin il y a une application biunivoque entre le fichier et son nom. Il est donc impossible de donner plusieurs noms à un même fichier. Ainsi pour créer une bibliothèque de fichiers 'TEXT' on est obligé de créer un nouveau fichier dans lequel on recopie tous les 'TEXT': il y a donc duplication d'information.

4.3) Aspect relationnel du catalogue.

4.3.1) Notion d'objet

Dans cette approche on essaie de distinguer les éléments qu'on manipule, et les liens qui éxistent entre ces éléments. Par définition on appellera objet l'élément que l'utilisateur peut manipuler dans sa totalité. Ainsi lorsqu'on parle de fichier il y a plusieurs objets: le contenu du fichier, le nom du fichier, les privilèges d'accès etc... Remarquons qu'au niveau du catalogue on considère le fichier comme un objet, c'est à dire comme un tout et on fait abstraction de la notion d'enregistrement qui est connue des méthodes d'accès seulement.

4.3.2) Nom historique.

Parmi les différents objets il y en a qui sont simples à manipuler et facilement distinguables entre eux. Par exemple les objets 'nom' sont en général des chaines de caractères relativement courtes (8 caractères). Il est donc aisé de distinguer deux noms entre eux et le système nommera les noms par les chaines de caractères elles-mêmes (ou par un système de codage pour minimiser la place mémoire occupée).

Par contre si on considère les objets fichiers, ils sont souvent de grande taille, ils résident sur support externe, il est donc impossible au système de les manipuler directement dans le catalogue. Le système donnera un nom unique à chaque fichier c'est à dire un nom historique. Ceux-ci peuvent être obtenus à l'aide d'un compteur, un TIMER etc... Dans le catalogue on ne conservera que le nom historique à l'aide duquel le système sera capable de retrouver la description du fichier. Cette technique vient du fait qu'il n'éxiste pas de moyens efficaces de manipuler un fichier directement d'aprés son contenu. Le système part du principe que tous les fichiers sont différents même si effectivement il y en a qui ont le même contenu.

D'autre part les objets crées par le système, qui sont transparents à l'utilisateur, seront aussi nommés à l'aide de noms historiques.

4.3.3) Relation

Par définition nous appellerons classe d'objets, notée C, un ensemble d'objets d'un même type. Par exemple on distinguera: la

classe des noms, la classe des privilèges d'accès etc...

Une relation est par définition un lien entre des classes d'objets:

$$R(C1,C2,C3, \ldots ,Cn)$$

c'est à dire un ensemble de n-uplets:

(c1i,c2j, ... ,cnk) avec cpi$\in$Cp pour tout i et p=1,2, ... ,n

Chaque classe d'objet représente un champ. Pour que la relation soit parfaitement définie, il faut qu'il y ait au moins un champ dont les objets soient tous différents. Ceci est toujours possible si on enlève les n-uplets identiques.

4.3.4) Représentation du catalogue.

Si on désire représenter le catalogue en arbre du 4.2 il suffit de définir cinq classes d'objets:

- les noms
- les utilisateurs
- les privilèges d'accès
- les noeuds
- les fichiers

Le catalogue est défini par la relation:

Rc(nom,noeud,noeud père)

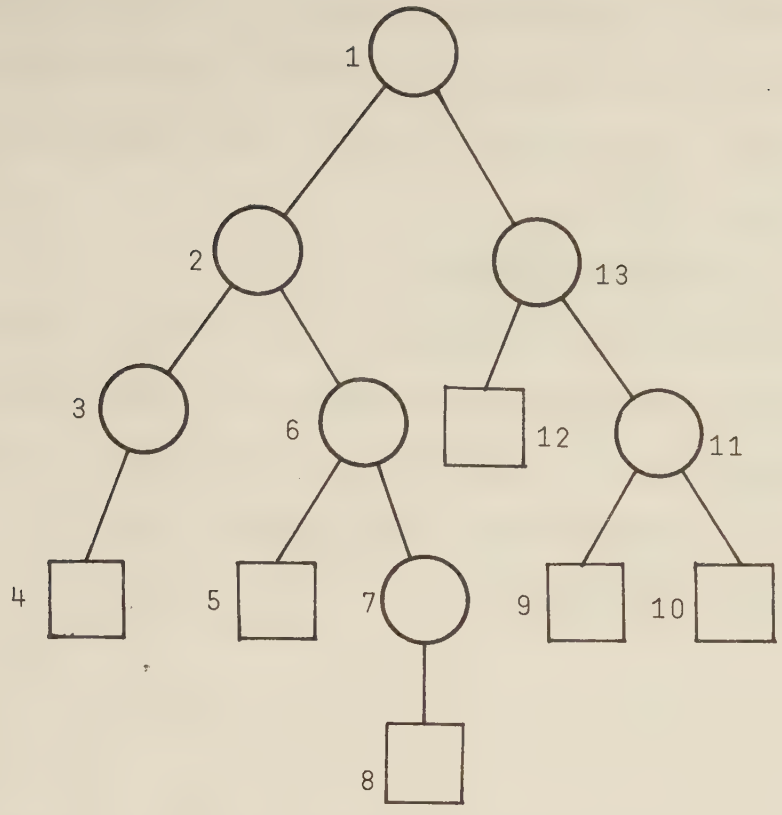
qui définit logiquement un arbre, alors que de façon interne on peut représenter la suite des triplets (cf fig. 4.9).

Le problème des listes d'accès peut se résoudre par une relation donnant les pouvoirs d'un utilisateurs:

Rp(utilisateur,noeud,privilège d'accès)

On constate qu'il est possible de séparer les représentations des relations en mémoire ce qui facilite leur gestion.

L'intérêt de cette méthode est de laisser au système ou à



Aspect logique

Représentation

Relation Rc

A	1	0
B	2	1
C	3	2
D	4	3
E	13	1
F	6	2
G	5	6
H	7	6
D	8	7
J	12	13
K	11	13
L	9	11
M	10	11

Figure 4.9 Relation catalogue

l'utilisateur la possiblité de créer des relations ce qui donne une grande souplesse d'emploi. Si on reprend l'exemple des 'TXTLIB' de CMS obtenues en recopiant des fichiers 'TEXT', ce même processus peut être réalisé avec une relation:

Rt(nom,fichiers)

qui exprime que des fichiers sont liés sous le nom de la 'TXTLIB'. On évite de dupliquer l'information d'une part, et d'autre part la modification des composants de la 'TXTLIB' peut se faire à l'aide des fonctions classiques de modification des fichiers alors que sous CMS il faut des fonctions spéciales.

4.4) Conclusion.

Il est certain que le catalogue est un des maillons principaux de la gestion de fichiers. C'est de lui que dépend une bonne partie des possibilités offertes à l'utilisateur. L'approche relationnelle est séduisante mais les problèmes de représentation en mémoire sont nombreux. Le résultat de tests permettra de faire un choix.

C O N C L U S I O N

Nous avons étudié et réalisé en partie une gestion de fichiers organisée en niveaux hiérarchiques. Le fait d'utiliser un système tel que CP/CMS a permis de bien séparer le monde réel d'une part connu de CP et le monde virtuel. Ceci a contribué à définir le concept d'environnement c'est à dire de faire accéder une machine virtuelle à certaines fonctions de CP (pagination par exemple). De cette manière on a pu créer des unités périphériques 'logiques' entièrement supportées par CP qui constituent le premier niveau de la gestion de fichiers de CMS+ (1-ESPACE) et qui rendent ce dernier indépendant du monde réel. Une telle propriété nous parait fondamentale pour une gestion de fichiers.

L'expérience faite sous CMS et les mesures qui ont été prises ont montré l'intérêt des 1-ESPACE en améliorant le rendement global du système surtout à forte charge. Ceci présage donc de bonnes performances lorsque tous les fichiers des utilisateurs seront traités par ces mécanismes. Nous avons atteint les objectifs que nous nous étions fixés à savoir créer un niveau indépendant, et améliorer les performances.

En ce qui concerne les 2-ESPACE, la réalisation est en cours. Le

choix fait de séparer entièrement le contenu et la description nous parait fondamentale. Les principaux problèmes viennent de la réalisation du segment descripteur et du catalogue. L'aspect relationnel de celui-ci est intéressant mais la représentation en mémoire des relations est possible sous diverses formes et actuellement des tests sont en cours pour avoir une idée des performances. Un autre point délicat est l'intégration dans le catalogue du partage des fichiers.

La suite logique de cette thèse est la définition et la réalisation du niveau supérieur: les 3-ESPACE c'est à dire les méthodes d'accès. Jusqu'à présent on s'est contenté d'accéder aux enregistrements de longueur fixe (séquentiel, direct), de longueur variable en plaçant la longueur de l'enregistrement dans le fichier lui-même ce qui est contraire au choix que nous avons fait de séparer le fichier et sa description.

Le plus gros problème est de définir des méthodes d'accès par clés. Ceci revient à considérer le 2-ESPACE comme une mémoire associative et d'y accéder d'aprés son contenu. Remarquons que ces méthodes permettraient de traiter les fichiers avec enregistrements de longueur variable en prenant comme clé le numéro de l'enregistrement.

La solution que nous pensons adopter est de définir un fichier comme un doublet:

(2-ESPACE,INDEXE)

où le 2-ESPACE contient les enregistrements, et l'INDEXE est un fichier particulier qui contient les clés et les adresses des enregistrements, et qui est éventuellement vide avec les méthodes d'accès classiques. L'INDEXE a une structure différente de celle des 2-ESPACE et n'est pas accessible à l'utilisateur. Nous pensons adopter une structure analogue à celle employée par OS/VSAM (I9) qui

organise les clés en un arbre équilibré (autant de branches à chaque noeud). Les feuilles décrivent les enregistrements dans le 2-ESPACE et les noeuds décrivent les noeuds ou feuilles des niveaux inférieurs.

Enfin nous espèrons que la gestion de fichiers du système CP+/CMS+, loin d'être parfaite, contribuera à améliorer le système actuel en donnant plus de possibilités à l'utilisateur ce qui est le principal objectif d'un système. D'autre part les mécanismes réalisés (1-ESPACE, 2-ESPACE) sont suffisement généraux pour servir de base à d'autres réalisations: 'data base' par exemple.

B I B L I O G R A P H I E

A1 J.R. ABRIAL, J. BAS, G. BEAUME,
G. HENNERON, R. MORIN, G. VIGLIANO
'Projet SOCRATE'

A2 A. AUROUX, C. HANS
'Notion de machines virtuelles'
Revue de l'AFIRO , Décembre 68 No 15

A3 A. AUROUX
'Généralisation du concept d'espace d'adressage. Application au système CP-67'
Congrés de l'AFCET novembre 72

A4 B.W. ARDEN, B.A. GALLER
T.C. O'BRIEN, F.H. WESTERVELD
'Program and addressing structure in a time-sharing environnement'
ACM 13,1 janvier 66

D1 P.J. DENNING
'Virtual memory'
Computing Surveys vol 2 No 3 septembre 70

D2 R.C. DALEY, P.G. NEUMANN
'A general purpose file system for secondary storage'
Proc FJCC 1965

D3 R.C. DALEY, J.B. DENNIS
'Virtual memory processes and sharing in MULTICS'
ACM 11,11 Mai 68

D4 E.W. DJIKSTRA
'The structure of 'THE' multiprogramming system'
ACM 11,5 Mai 68

G1 GRENOBLE SYSTEM LANGUAGE
IBM Grenoble scientific center
Report No FF2.0133, janvier 72

H1 C. HANS, J.P. LE HEIGET
'Généralisation de la notion d'espace virtuel sous le système CP-67'
Congrés de l'AFCET novembre 72

I1 IBM Controle program 67/ Cambridge monitor system
User guide
GH20-0859

I2 IBM Control program 67
Program logic manual
GY20-0590

I3 IBM System 360/67
Functional characteristic
GA27-2719

I4 IBM System 360
Principles of operation
A22-6821

I5 IBM TSS/360
System logic summary
GY28-2009

I6 IBM OS/360
ISAM program logic manual
GY28-6618

I7 IBM TSS/360
Data management facilities
GC28-2056

I8 IBM System 370
Principles of operation
GA22-7000

I9 IBM OS/VS
Virtual storage acces method
GC26-3799-0

I10 IBM VIRTUAL MACHINE FACILITY/370
Command language, User's Guide
GC20-1804-0

K1 D.E. KNUTH
'The art of computer programming'
Addison-Wesley, Reading Mass 1968

K2 R.M. KOGUT
'The Segment Based File Support System'
ACM Workshop on Virtual Computer Systems, mars 1973

L1 J.P. LE HEIGET
'Généralisation de la notion d'espace virtuel sous les systèmes CP-67/CMS'
Thèse C.N.A.M. juillet 1972

L2 P. LEFEBVRE
'SPY: Un système de contrôle pour la mise au point de systèmes de programmation grâce à un couplage de machines virtuelles'
Université de Grenoble, Thèse juillet 1972

L3 J. LEROUDIER
'Une analyse de système'
Université de Grenoble, Thèse avril 1973

L4 D. LEFKOVITZ
'File structures for on-line systems'
Spartan books

L5 X. de LAMBERTERIE
'Unités périphériques logiques (1-ESPACE) sous CP'
Séminaire à l'I.M.A.G. mars 73

M1 S.E. MADNICK, J. ALSOP
'A modular approach to file system design'
SJCC 1969

M2 J.R. MARTINSON
'Utilization of virtual memory in TSS/360'
IBM syst. TR 53.0001

O1 E.I. ORGANICK
'The MULTICS system: an examination of its structure'
M.I.T. Press

S1 H. SAVARY
'Etude et réalisation d'un système de mise au point assisté'
Université de Grenoble, Thèse à paraitre

R1 J. RODRIGUEZ-ROSSEL, J.P. DUPUY
'The evaluation of a time-sharing demand system'
SJCC 1972

W1 R.W. WATSON

'Time sharing design concept'

Mac Graw-hill

T A B L E D E S
M A T I E R E S

CHAPITRE 1

1) HOMOGENEISATION DES UNITES PERIPHERIQUES REELLES 2

2) LA GESTION DE FICHIERS DE CMS+ 7

CHAPITRE 2: 1-ESPACE

1) CONCEPT DE 1-ESPACE 11

1.1) Définition 11

1.2) Caractéristiques 12

1.3) Relations avec la notion de mini-disque 16

1.4) Application dans l'espace physique 19

2) LE 1-ESPACE VU PAR L'UTILISATEUR 21

2.1) Aspect externe 21

2.2) Réconnaissance d'un 1-ESPACE 21

2.3) Lecture 22

2.4) Ecriture 22

2.5) Test des pages modifiées 22

3) PRINCIPE DE REALISATION SOUS CP 24

3.1) Définition dans le catalogue 24

3.2) Activation d'un 1-ESPACE par CP 25

3.3) Primitives de manipulation 30

3.4) Désactivation d'un 1-ESPACE 35

4) UN EXEMPLE D'EMPLOI: LE S-ESPACE DE CMS 36

4.1) Rappels sur la gestion de fichiers de CMS 36

4.2) Rappels sur le disque système de CMS 38

4.3) Le S-ESPACE 39

4.4) Création du S-ESPACE 40

4.5) Transfert du S-ESPACE sur tambour 41

4.6) Acquisation par l'utilisateur 43

4.7) Emploi des S-dules 43

5) MESURES 45

5.1) Environnement de mesure 45

5.2) Aspect externe des mesures 45

5.3) Aspect interne des mesures 46

5.4) Conclusion 56

CHAPITRE 3: M-ESPACE

1) DEFINITION 63

2) CARACTERISTIQUES 64

2.1) Partage de la mémoire 64

2.2) Exploitation de la segmentation 66

2.3) Segment entité logique 68

2.4) Protection de la mémoire 69

2.5) Croissance dynamique des segments 70

3) APPLICATION DANS L'ESPACE PHYSIQUE 72

4) REALISATION SOUS CP 74

4.1) Définition dans le catalogue 74

4.2) Activation des M-ESPACE par CP 74

5) EXTENSIONS DES MECANISMES 76

5.1) Croissance dynamique des segments 76

5.2) Protection au niveau du segment 77

5.3) Partage des segments 78

CHAPITRE 4: 2-ESPACE

1) CONCEPT DE 1-ESPACE 81

1.1) Définition 81

1.2) Caractéristiques 82

1.3) Traduction des adresses 86

2) LE 2-ESPACE VU PAR L'UTILISATEUR 88

2.1) 2-ESPACE mémoire virtuelle 88

2.2) Primitive de lecture 88

2.3) Primitive d'écriture 88

3) PRINCIPE DE REALISATION 89

3.1) Contexte de réalisation 89

3.2) Structure du contenu d'un 1-ESPACE 89

3.3) Le segment descripteur 91

3.4) Opérations sur un 2-ESPACE 109

3.5) Opérations sur un 1-ESPACE 112

4) LE CATALOGUE 116

4.1) Aspect externe 116

4.2) Réalisation sous forme d'arbre 118

4.3) Aspect relationnel du catalogue 120

4.4) Conclusion 124

CONCLUSION 125

BIBLIOGRAPHIE 128